

[illegible][illegible]

```
PPPPPPPP  UU      UU  DDDDDDDD  RRRRRRRR  IIIIII  VV      VV  EEEEEEEEE  RRRRRRRR
PPPPPPPP  UU      UU  DDDDDDDD  RRRRRRRR  IIIIII  VV      VV  EEEEEEEEE  RRRRRRRR
PP      PP  UU      UU  DD      DD  RR      RR  II      II  VV      VV  EE      EE  RR      RR
PP      PP  UU      UU  DD      DD  RR      RR  II      II  VV      VV  EE      EE  RR      RR
PP      PP  UU      UU  DD      DD  RR      RR  II      II  VV      VV  EE      EE  RR      RR
PP      PP  UU      UU  DD      DD  RR      RR  II      II  VV      VV  EE      EE  RR      RR
PPPPPPPP  UU      UU  DD      DD  RRRRRRRR  II      II  VV      VV  EEEEEEEE  RRRRRRRR
PPPPPPPP  UU      UU  DD      DD  RRRRRRRR  II      II  VV      VV  EEEEEEEE  RRRRRRRR
PP      UU      UU  DD      DD  RR      RR  II      II  VV      VV  EE      EE  RR      RR
PP      UU      UU  DD      DD  RR      RR  II      II  VV      VV  EE      EE  RR      RR
PP      UU      UU  DD      DD  RR      RR  II      II  VV      VV  EE      EE  RR      RR
PP      UU      UU  DD      DD  RR      RR  II      II  VV      VV  EE      EE  RR      RR
PP      UU      UU  DD      DD  RR      RR  II      II  VV      VV  EE      EE  RR      RR
PP      UU      UU  DD      DD  RR      RR  II      II  VV      VV  EE      EE  RR      RR
UUUUUUUUUU  DDDDDDDD  RR      RR  IIIIII  VV      VV  EEEEEEEEE  RR      RR
UUUUUUUUUU  DDDDDDDD  RR      RR  IIIIII  VV      VV  EEEEEEEEE  RR      RR
                                         ....
                                         ....
                                         ....
                                         ....
```

```
LL      IIIIII  SSSSSSSS
LL      IIIIII  SSSSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SSSSSS
LL      II      SSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LLLLLLLLLL  IIIIII  SSSSSSSS
LLLLLLLLLL  IIIIII  SSSSSSSS
```


(2)	320	DEFINITIONS
(2)	356	+ UDA Command Packet Layout
(2)	535	+ Define PU Port specific PDT extension
(2)	699	+ Define PU specific UCB extension
(2)	830	DRIVER STRUCTURES
(2)	831	+ Driver Prologue Table
(2)	865	+ Driver Dispatch Table
(2)	924	+ "Register" Dump routine
(2)	961	+ Function Decision Table
(2)	976	+ Static Storage
(2)	996	+ NULL MESSAGE AND DATAGRAM INPUT ROUTINES
(2)	997	+ NULL CDT
(2)	1035	Request and Release DATAPATH transfer vectors
(2)	1131	INITIALIZATION
(2)	1198	+ UNIT INIT
(2)	1230	+ CONTROLLER INIT
(2)	1393	+ Build PDT
(2)	1645	+ TRACE COMMAND and TRACE_RESPONSE
(2)	1699	+ INIT_UDA BUFFERS
(2)	1983	+ VIRT_TO_PHYAD
(2)	2015	+ INIT_INIT STEPS
(2)	2143	+ Hardware Initialization
(2)	2556	+ INIT_PU_PDT Fill in variable
(2)	2557	+ PDT data
(2)	2664	+ BUILD_PB_SB Build System Block and Path Block
(2)	2783	+ UPDATE_PB_SB
(2)	2812	+ ALLOC_POOL
(3)	2873	UNIMPLEMENTED FORK PROCESS CALLS
(4)	2894	MRESET and MSTART
(4)	2919	CONNECTION MANAGEMENT CALLS
(4)	2920	+ FPC\$CONNECT, COMPLETE PROCESSING A CONNECT
(5)	3086	+ FPC\$DISCONNECT, PROCESS A DISCONNECT CALL
(5)	3139	SEQUENCED MESSAGE CALLS
(5)	3140	+ FPC\$ALLOCMSG, ALLOCATE A MESSAGE BUFFER
(5)	3219	+ FPC\$DEALLOCMSG, DEALLOCATE A MESSAGE BUFFER
(5)	3308	+ FPC\$RCHMSGBUF, RECYCLE MESSAGE BUFFER
(5)	3309	+ AT HIGH PRIORITY
(5)	3310	+ FPC\$RCLMSGBUF, RECYCLE MESSAGE BUFFER
(5)	3311	+ AT LOW PRIORITY
(5)	3377	+ FPC\$SND CNTMSG, SEND COUNTED SEQUENCED MESSAGE
(5)	3477	+ FPC\$MAPIRP, Map a buffer
(5)	3529	+ FPC\$MAPIRP UV1, Map a buffer for uVAX I
(5)	3580	+ MAP UNALIGN uVAX I Q-BUS Map Unaligned Buffer
(5)	3667	SETUP_COPY_SEG1 and SETUP_COPY_SEG2
(5)	3782	+ FPC\$MAPIRP_BDA, Map a Buffer for BDA
(5)	3791	+ FPC\$UNMAP, Release mapping resources
(5)	3830	+ FPC\$UNMAP UV1, Release mapping resources
(5)	3907	+ FPC\$UNMAP_BDA, Release mapping resources BDA
(5)	3916	INTERNAL SUBROUTINES
(5)	3917	+ POLL CMDRING
(5)	4017	- STATE_ERR, RETURN CDT STATE ERROR
(5)	4018	- TO SYSAP
(5)	4031	INTERRUPT SERVICING
(5)	4032	+ PUSINT - Interrupt service routine
(5)	4086	+ POLL RSPRING
(5)	4248	+ PU\$SA_POLL

```
0000 1      .TITLE PUDRIVER.
0000 2      .IDENT 'V04-000'
0000 3
0000 4      *****
0000 5      *
0000 6      *  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0000 7      *  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0000 8      *  ALL RIGHTS RESERVED.
0000 9      *
0000 10     *  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0000 11     *  ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0000 12     *  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0000 13     *  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0000 14     *  OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0000 15     *  TRANSFERRED.
0000 16     *
0000 17     *  THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 18     *  AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 19     *  CORPORATION.
0000 20     *
0000 21     *  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 22     *  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0000 23     *
0000 24     *
0000 25     *****
0000 26
0000 27     ++
0000 28
0000 29     FACILITY:
0000 30
0000 31     VAX/VMS EXECUTIVE, I/O DRIVERS
0000 32
0000 33     ABSTRACT: This module contains the UDA port driver.
0000 34
0000 35     AUTHORS: Richard I. Hustvedt, July 1981
0000 36             Robert Rappaport
0000 37
0000 38     MODIFIED BY:
0000 39
0000 40     V03-159 RLRSTEP4      Robert L. Rappaport      16-Jul-1984
0000 41             Expand CNTRLTYP field in Port Step 4 from 4 bits to 7 bits.
0000 42             Also add in Scorpio-BUA support and hooks for BDA support.
0000 43
0000 44     V03-158 RLRQDA      Robert L. Rappaport      19-Jun-1984
0000 45             Add recognition of QDA.
0000 46
0000 47     V03-157 RLRTTRACE   Robert L. Rappaport      01-Jun-1984
0000 48             Add support to be able to dynamically configure tracing.
0000 49
0000 50     V03-156 RLRMAYA      Robert L. Rappaport      22-May-1984
0000 51             Add in MAYA tape support. Also add in ability to just
0000 52             trace "PT" ports.
0000 53
0000 54     V03-155 RLRMVER      Robert L. Rappaport      26-Apr-1984
0000 55             Add code to send operator message to complain about ucode
0000 56             out of date on RDRX controller.
0000 57
```


0000 58 :
0000 59 :
0000 60 :
0000 61 :
0000 62 :
0000 63 :
0000 64 :
0000 65 :
0000 66 :
0000 67 :
0000 68 :
0000 69 :
0000 70 :
0000 71 :
0000 72 :
0000 73 :
0000 74 :
0000 75 :
0000 76 :
0000 77 :
0000 78 :
0000 79 :
0000 80 :
0000 81 :
0000 82 :
0000 83 :
0000 84 :
0000 85 :
0000 86 :
0000 87 :
0000 88 :
0000 89 :
0000 90 :
0000 91 :
0000 92 :
0000 93 :
0000 94 :
0000 95 :
0000 96 :
0000 97 :
0000 98 :
0000 99 :
0000 100 :
0000 101 :
0000 102 :
0000 103 :
0000 104 :
0000 105 :
0000 106 :
0000 107 :
0000 108 :
0000 109 :
0000 110 :
0000 111 :
0000 112 :
0000 113 :
0000 114 :

- V03-154 RLRPDTADP Robert L. Rappaport 9-Apr-1984
1) Init PDT\$L ADP. 2) Initialize PDT\$L WAITQFL list head
in INIT PU PDT. 3) Clear connection active bit in
PDT\$B_CONBITMAP on calls to disconnect.
- V03-153 ROW0334 Ralph O. Weber 3-APR-1984
Add setup of PDT\$L_MAXBCNT to BUILD_PDT. Give PDT\$L_MAXBCNT a
value of 127*512 (i.e. 127 blocks).
- V03-152 RLRPHYPGS Robert L. Rappaport 21-Mar-1984
Make use of new system routine to allocate physically
contiguous pages. Also add null routine FPC\$STOP_VCS.
- V03-151 PRD0070 Paul R. DeStefano 25-Feb-1984
Clear SB\$S_CSB (link to newest CSB) when system block
is created.
- V03-150 RLRDELPQ Robert L. Rappaport 25-Jan-1984
Eliminate separate conditionalized PQDRIVER and rather
use CPUDISP to accomodate differences. In general,
differences are in initialization code or in main line code.
For initialization code merely use CPUDISP. For main line
code (MAPIRP and UNMAP), have two sets of entry points,
one for all CPU's except uVAX I, and one set for uVAX I.
Then at initialization time, in BUILD_PDT, build PDT
dispatch table to dispatch to proper entry point for the
processor we are running on.
- Also fix bug in uVAX I BUILD_PDT that manifested itself when
we have more than one port; namely we tried to map the
map registers in the 2nd PDT into the System Addresses
pointed at by @ADP\$S_CSR+^x800 of the common ADP, where we
had already mapped the map registers from the first PDT.
Solution is to only go thru map code once. A new flag,
MAP\$V_MAPREGS, in PUSL_DRIVER_STS, if set, means that the code
has already been executed.
- V03-149 ROW0268 Ralph O. Weber 28-DEC-1983
Change instructions used to set aside space for NULL_CDT so
that space allocated is always large enough to accomodate a
complete CDT. (I.E. base the space allocation on CDT\$K_LENGTH.)
- V03-148 RLREXEALLOC Robert L. Rappaport 22-Dec-1983
Correct subtle bug introduced in previous fix. Namely
the call to EXE\$ALONONPAGED was restored and the call to
EXE\$ALLOCATE was removed. To deal with the problem of
deallocating fragments that might be in LRP space,
the deallocation of the PDT fragment (in PQDRIVER BUILD_PDT)
was eliminated and it was decided to forget about that small
bit of space.
- V03-147 RLRCRED Robert L. Rappaport 18-Nov-1983
Bring driver into conformance with latest UQPORT spec.
1. Ignore credits field (in envelope) for Maintenance
type messages.
2. Wait for at least 100 uSecs after initialization
interrupts before reading SA to look for step bit.

3. Test for controller tolerance of odd addresses before segmenting a transfer and using the aligned buffer.

0000	115	:	
0000	116	:	
0000	117	:	
0000	118	:	
0000	119	:	
0000	120	:	
0000	121	:	
0000	122	:	
0000	123	:	
0000	124	:	
0000	125	:	
0000	126	:	
0000	127	:	
0000	128	:	
0000	129	:	
0000	130	:	
0000	131	:	
0000	132	:	
0000	133	:	
0000	134	:	
0000	135	:	
0000	136	:	
0000	137	:	
0000	138	:	
0000	139	:	
0000	140	:	
0000	141	:	
0000	142	:	
0000	143	:	
0000	144	:	
0000	145	:	
0000	146	:	
0000	147	:	
0000	148	:	
0000	149	:	
0000	150	:	
0000	151	:	
0000	152	:	
0000	153	:	
0000	154	:	
0000	155	:	
0000	156	:	
0000	157	:	
0000	158	:	
0000	159	:	
0000	160	:	
0000	161	:	
0000	162	:	
0000	163	:	
0000	164	:	
0000	165	:	
0000	166	:	
0000	167	:	
0000	168	:	
0000	169	:	
0000	170	:	
0000	171	:	

V03-146 RLRPQ02 Robert L. Rappaport 14-Nov-1983
Clear up problems in non-aligned transfers. In particular, make it so that we copy WRITE data to the aligned page BEFORE the I/O and we copy READ data from the aligned page AFTER the I/O operation. We accomplish this by adding two subroutines, SETUP_COPY_SEG1 and SETUP_COPY_SEG2, that perform the necessary setup operations to prepare for a copy in either direction.

V03-145 KDM0104 Kathleen D. Morse 20-Oct-1983
Fix reference to MMGSGL_SPTBASE to be PIC.

V03-144 KDM0103 Kathleen D. Morse 01-Oct-1983
Invalidate virtual address after changing the system page table entry to do unaligned transfers. Allocate a system page table entry in the initialization routine for the Qbus.

V03-143 NPK3050 N. Kronenberg 30-Sep-1983
For PQ allocate PDT so that PDT\$PQ_MAP within the PDT is page aligned.

V03-142 KDM0102 Kathleen D. Morse 29-Sep-1983
Change Qbus code to compute physical address of ring buffer during initialization.

V03-141 KDM0101 Kathleen D. Morse 29-Sep-1983
Fix indexing through page table entries for Qbus.

V03-140 KDM0100 Kathleen D. Morse 29-Sep-1983
Change the way the ADP points to the map registers for the MicroVAX I. Fix virtual to physical translation. Add MicroVAX I to CPUDISP macros.

V03-139 CWH3139 CW Hobbs 17-Sep-1983
Change DT\$_RC25 symbol for Aztec port to DT\$_LESI so that DT\$_RC25 can refer to the removable pack on the Aztec.

V03-138 RLRNEWPB Robert L. Rappaport 28-Jul-1983
Incorporate new PBDEF changes.

V03-137 RLRQIOCHNLa Robert L. Rappaport 15-Jul-1983
Must refresh R4 => PDT after REQPCAN.

V03-136 RLRQIOCHNL Robert L. Rappaport 6-Jul-1983
Correct subtle error in QIO routine. After REQCOM, PU channel was released. Next interrupt crashed system. Fix is to have STARTIO, BSBW to REQCOM, so as to retain control after REQCOM. In this way we can REQPCAN again.

V03-135 RLRSAPOLL Robert L. Rappaport 5-Jul-1983

0000 172 :
0000 173 :
0000 174 :
0000 175 :
0000 176 :
0000 177 :
0000 178 :
0000 179 :
0000 180 :
0000 181 :
0000 182 :
0000 183 :
0000 184 :
0000 185 :
0000 186 :
0000 187 :
0000 188 :
0000 189 :
0000 190 :
0000 191 :
0000 192 :
0000 193 :
0000 194 :
0000 195 :
0000 196 :
0000 197 :
0000 198 :
0000 199 :
0000 200 :
0000 201 :
0000 202 :
0000 203 :
0000 204 :
0000 205 :
0000 206 :
0000 207 :
0000 208 :
0000 209 :
0000 210 :
0000 211 :
0000 212 :
0000 213 :
0000 214 :
0000 215 :
0000 216 :
0000 217 :
0000 218 :
0000 219 :
0000 220 :
0000 221 :
0000 222 :
0000 223 :
0000 224 :
0000 225 :
0000 226 :
0000 227 :
0000 228 :

Implement periodic polling of the SA register. Also
cleanup some miscellaneous bugs:

1. In HARDWARE_INIT, insert BRB 190\$ at the
end of the logic following label TESTUDA_780.
This corrects a bug that was permanently
wasting a buffered datapath on 780's.
2. Add a FUNCTAB +EXESZEROPARM for the
STOP and INITIALIZE qio functions that we now
support.
3. In INIT UDA_BUFFERS, save values placed into
UCB\$W_B0FF, UCB\$W_BCNT, and UCB\$S_SVAPTE in
new UCB fields. Then in HARDWARE_INIT, restore
these values immediately prior to the LOADUBA
invocation. This corrects the problem that
the QIO functions cause the these fields to
be cleared.

V03-134 RLRSRVCN1 Robert L. Rappaport 3-Jun-1983
Corect two bugs introduced in previous edit.

V03-133 RLRSRVCON Robert L. Rappaport 1-Jun-1983
1. Prevent logging redundant Initialization Log entries.
2. Correct infinte loop typo in Maintenance Type messages.
3. Connect changes:
a) Add PDT\$B_SERVERS, bit map that lists servers
supported at this port.
b) Have CONNECT return \$\$\$_FAILRSP if caller is
trying to Connect to a server not supported
on this port.
c) Have CONNECT return R2=>Connect data.

V03-132 RLRCPU DISP Robert L. Rappaport 25-May-1983
Use new form of CPUDISP macro.

V03-131 RLRPCHAR Robert L. Rappaport 20-May-1983
Set PDT\$M_SNGLHOST bit in PDT\$W_PORTCHAR field.

V03-130 RLRPUR780 Robert L. Rappaport 26-Apr-1983
Prevent losing context of which datapath to purge.
Do this by saving and then restoring the contents
of CRB\$S_INTD+VECSB_DATAPATH when doing a UDA
requested purge at device interrupt level.

V03-029 RLRUDAREV Robert L. Rappaport 11-Apr-1983
Test for out of rev UDA50's and UDA50A's on 780 systems.

V03-028 RLRSVAVCRED Robert L. Rappaport 8-Apr-1983
On Disconnect, save available credits in PDT\$W_PU_CREDx,
where x (ID) is index of disconnecting connection.

V03-027 RLRPQ01a Robert L. Rappaport 31-Mar-1983
Fix branch out of range brought about by previous addition
which also included setting UCB\$B_DEVTYPE according to
UQPORT type.

V03-026 RLRPQ01 Robert L. Rappaport 17-Mar-1983
Add conditionally assembled support for Q-BUS port.

0000 229 :
0000 230 :
0000 231 :
0000 232 :
0000 233 :
0000 234 :
0000 235 :
0000 236 :
0000 237 :
0000 238 :
0000 239 :
0000 240 :
0000 241 :
0000 242 :
0000 243 :
0000 244 :
0000 245 :
0000 246 :
0000 247 :
0000 248 :
0000 249 :
0000 250 :
0000 251 :
0000 252 :
0000 253 :
0000 254 :
0000 255 :
0000 256 :
0000 257 :
0000 258 :
0000 259 :
0000 260 :
0000 261 :
0000 262 :
0000 263 :
0000 264 :
0000 265 :
0000 266 :
0000 267 :
0000 268 :
0000 269 :
0000 270 :
0000 271 :
0000 272 :
0000 273 :
0000 274 :
0000 275 :
0000 276 :
0000 277 :
0000 278 :
0000 279 :
0000 280 :
0000 281 :
0000 282 :
0000 283 :
0000 284 :
0000 285 :

- V03-025 TCM0001 Trudy C. Matthews 28-Feb-1983
Update occurrences of CPUDISP macro so that the function correctly on an 11/790. In both cases, we just take the same code path as the 11/780.
- V03-024 RLRPBSB Robert L. Rappaport 11-Feb-1983
Cleanup minor phasing problem in Path Block and System Block initialization by adding new subroutine, UPDATE_PB_SB.
- V03-023 RLMSGTYP Robert L. Rappaport 3-Feb-1983
Add ability to handle simple credit type messages and maintenance messages.
- V03-022 RLRDUP1 Robert L. Rappaport 31-Jan-1983
Fix typo in original RLRDUP fix that used R5 instead of R0.
- V03-021 RLRLUSECNT Robert L. Rappaport 25-Jan-1983
Modify logic that permanently allocates Buffered Data Path on VAX-11/750. From now on we will only permanently allocate a BDP if the controller is a UDA. Other UQPORT controllers "semi-permanently" allocate a BDP for the duration of a burst. This is implemented via adding a new cell, PDT\$B_BDPUSECNT, which counts the number of transfers in a burst that are currently using the "semi-permanent" BDP. When this count goes to zero, the BDP is deallocated. For the UDA, the usecount is biased by one, so that it never goes to zero and therefore never get deallocated.
- V03-020 RLRPPFORK Robert L. Rappaport 7-Jan-1983
Eliminate Bugcheck in POST_POWER_FORK, that was activated when it found the UCB fork block busy. We do this by defining a new UCBSW_DEVSTS flag, UCBSM_PU_MRESET, that if set causes PUDRIVER to reset itself upon awakening from the busy UCB.
- V03-019 RLRLDUP Robert L. Rappaport 5-Jan-1983
Add support for two QIO functions that (1) effectively shut off Class Drivers from the port and only let DUP connections thru, and then (2) reopen the port. The Shutoff QIO function is IOS_STOP and the reopen function is IOS_INITIALIZE.
- V03-018 RLRCREDITa Robert L. Rappaport 20-Dec-1982
Additional correction to take care of bug introduced by original fix. Must add space to NULL CDT to accomodate CDT\$L_CRWAITQFL and CDT\$L_CRWAITQBL, and also code to initialize this header.
- V03-017 RLRCREDIT Robert L. Rappaport 3-Dec-1982
Corrected bug brought out in TUB1 testing where the credit that was received on the End Packet of a GET COMMAND STATUS (i.e. the "immediate" credit) was assigned to a waiter on the credit resource wait queue. Then the time out mechanism had no credits to allocate. The fix is to add in the


```
0000 286 :
0000 287 :
0000 288 :
0000 289 :
0000 290 :
0000 291 :
0000 292 :
0000 293 :
0000 294 :
0000 295 :
0000 296 :
0000 297 :
0000 298 :
0000 299 :
0000 300 :
0000 301 :
0000 302 :
0000 303 :
0000 304 :
0000 305 :
0000 306 :
0000 307 :
0000 308 :
0000 309 :
0000 310 :
0000 311 :
0000 312 :
0000 313 :
0000 314 :
0000 315 :
0000 316 :
0000 317 :
0000 318 :--
```

received credits upon receipt of the message,
then call the Input Dispatcher and only then
after return try to resume waiters for credits.

V03-016 RLRTBUG Robert L. Rappaport 8-Oct-1982
Correct in trace when we have more than 1 UDA.

V03-015 RLRPATHB Robert L. Rappaport 6-Oct-1982
Put proper port name into path block. Also fill in
SBSQ_SWINCARN with value in EXESGQ_SYSTIME.

V03-014 RLRUDASA Robert L. Rappaport 12-Aug-1982
Read UDASA after IOFORK in HARDWARE_INIT so that TU81
has enough time to update UDASA after interrupting.

V03-013 KDM0002 Kathleen D. Morse 28-Jun-1982
Added \$DYNDEF, \$PRDEF, \$SSDEF, and \$VADEF.

V03-012 RLREC003 Robert L. Rappaport 4-June-1982
Add separate REQDATAP_750 and REQDATAP_730 to replace
REQDATAP_750730. New routine for 750 will only use
pre-allocated buffered datapath if the transfer is
longword aligned. Otherwise it will use direct datapath.

V03-011 RLREC002 Robert L. Rappaport 7-May-1982
INSQUE Path Block onto System Block.

V03-010 RLREC001 Robert L. Rappaport 7-May-1982
Add purging of UNIBUS Buffered Data Path in UNMAP.
This tracks patch made to V3.0 before release.

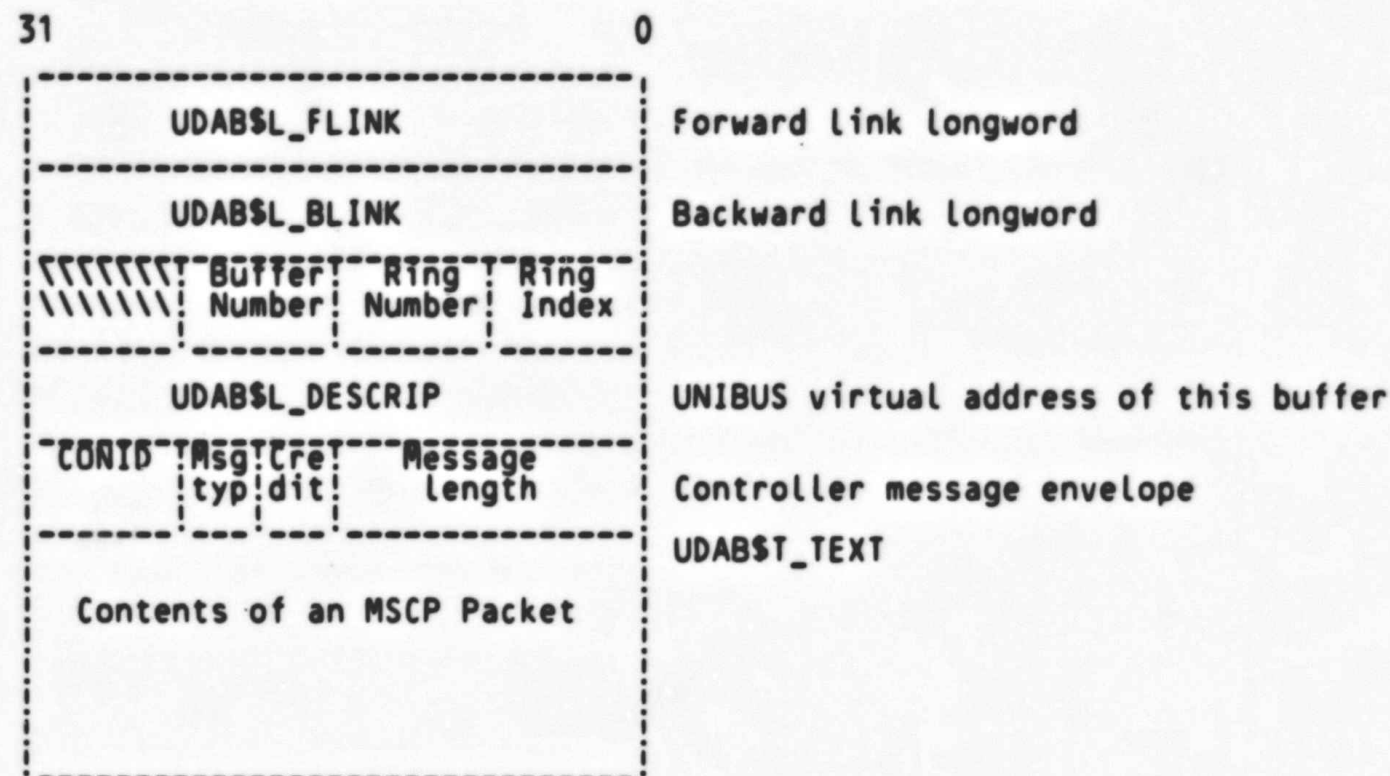
DEFINITIONS

0000	320	.SBTTL	DEFINITIONS	
0000	321	::		
0000	322	::	System definitions (LIB.MLB):	
0000	323	::		
0000	324			
0000	325	\$ADPDEF		:Adapter Control Block offsets
0000	326	\$CDRPDEF		:Class Driver Request Packet offsets
0000	327	\$CDTDEF		:Connection Descriptor offsets
0000	328	\$CRBDEF		:Channel Request Block offsets
0000	329	\$DCDEF		:Device type codes
0000	330	\$DDBDEF		:Device Data Block offsets
0000	331	\$DPTDEF		:Driver Prologue Table offsets
0000	332	\$DYNDDEF		:Dynamic data structure types
0000	333	\$EMBDEF		:Error log definitions
0000	334	\$EMBLTDEF		:Log message type codes
0000	335	\$IDBDEF		:IDB offsets
0000	336	\$IODEF		:IO function definitions
0000	337	\$IPLDEF		:IPL symbolic definitions
0000	338	\$IRPDEF		:IRP offsets
0000	339	\$MSGDEF		:Opcom message code definitions
0000	340	\$PBDEF		:Path Block offsets
0000	341	\$PDTDEF		:Port Descriptor Table offsets
0000	342	\$PRDEF		:Processor register numbers
0000	343	\$PTEDEF		:Page Table Entry definitions
0000	344	\$RDDEF		:Response Descriptor offsets
0000	345	\$SBDEF		:System Block offsets
0000	346	\$SCSCMGDEF		:SCS Connection Management format
0000	347	\$SDIRDEF		:SCS Directory Entry offsets
0000	348	\$SSDEF		:System status codes
0000	349	\$UBADEF		:Unibus Adapter registers
0000	350	\$UBMDDEF		:Unibus Mapping Descriptor format
0000	351	\$UCBDEF		:Unit Control Block offsets
0000	352	\$VADEF		:Virtual address fields
0000	353	\$VECDEF		:CRB transfer vector blk offsets
0000	354			

+ UDA Command Packet Layout

.SBTTL + UDA Command Packet Layout

Each UDA command packet has the following structure where the individual fields are described below the diagram.



\$UDABDEF -- Define UDA packet buffer structure. Structure includes port driver header, controller header and text body. The port driver header contains:

1. A FLINK and a BLINK for queueing the buffer on the free queue and also on the SEND Q.
2. \$B_RINGINX which contains the index into a ring on which this buffer has been placed (valid only if the buffer is NOT on the free queue).
3. \$B_RINGNO which contains the number (0 => command ring and 1 => response ring) of the ring on which the buffer is currently residing.
4. \$B_BUFFNO which contains the number of this buffer. This value serves as an index into the PDTSL_BDTABLE, whose elements point to these buffers.
5. \$L_DESCRIP which contains the UNIBUS virtual address of the text portion of this buffer in the low order 30 bytes of this longword, and which also has the two high order bits (ownership and full bits) set.

0000 356
0000 357
0000 358 :
0000 359 :
0000 360 :
0000 361 :
0000 362 :
0000 363 :
0000 364 :
0000 365 :
0000 366 :
0000 367 :
0000 368 :
0000 369 :
0000 370 :
0000 371 :
0000 372 :
0000 373 :
0000 374 :
0000 375 :
0000 376 :
0000 377 :
0000 378 :
0000 379 :
0000 380 :
0000 381 :
0000 382 :
0000 383 :
0000 384 :
0000 385 :
0000 386 :
0000 387 :
0000 388 :
0000 389 :
0000 390 :
0000 391 :
0000 392 :
0000 393 :
0000 394 :
0000 395 :
0000 396 :
0000 397 :
0000 398 :
0000 399 :
0000 400 :
0000 401 :
0000 402 :
0000 403 :
0000 404 :
0000 405 :
0000 406 :
0000 407 :
0000 408 :
0000 409 :
0000 410 :
0000 411 :
0000 412 :

+ UDA Command Packet Layout

```

0000 461
0000 462 ; Define Device I/O Page Registers
0000 463
0000 464 $DEFINI UDA
0000 465 $DEF UDAIP .BLKW 1 ; Initialization and Polling Register
0002 466 $DEF UDASA .BLKW 1 ; Status, Address, & VAX Purge ACK Register
0004 467 $DEFEND UDA
0000 468
0000 469
0000 470 ; ++
0000 471 ; Local symbol definitions
0000 472 ;--
0000 473
0000FAB 0000 474 LOOP_LIMIT = ^X<FAB> ; Step 1 maximum wait time for response
000000B8 0000 475 INTR_VEC = ^O<270> ; Primary Interrupt vector
0000 476
00000000 0000 477 UDASK_SEQMSGTYP = 0 ; Sequential Message Type
00000001 0000 478 UDASK_DGTYPE = 1 ; Datagram type
00000002 0000 479 UDASK_CREDTYPE = 2 ; Credit type
0000000F 0000 480 UDASK_MAINTTYPE = 15 ; Maintenance type
00000004 0000 481 UDASK_RINGEXP = 4 ; Log base 2 of desired ring size
00000010 0000 482 UDASK_RINGSIZE = 1aUDASK_RINGEXP ; Number of Ring & Packet entries
0000 483
0000 484 ; Command and Message Ring Control Flags
0000 485
0000001E 0000 486 UDA_V_FLAG = 30 ; Buffer control bit number
40000000 0000 487 UDA_M_FLAG = 1 a UDA_V_FLAG ; Buffer control flag mask
0000001F 0000 488 UDA_V_OWN = 31 ; Own flag bit number
80000000 0000 489 UDA_M_OWN = 1 a UDA_V_OWN ; Own flag mask
0000 490
80000000 0000 491 UQPORT_M_MAPPED=1a31 ; Transfer Mapped by port.
0000 492
0000000A 0000 493 STEP1_LIMIT=10 ; Number of seconds for STEP 1 timeout
0000000A 0000 494 STEP2_LIMIT=10 ; Number of seconds for STEP 2 timeout
0000000A 0000 495 STEP3_LIMIT=10 ; Number of seconds for STEP 3 timeout
0000 496
00000080 0000 497 NUMUBAVEC=128 ; Number of slots in UBA vector
0000 498
00000005 0000 499 NO_CONSEC_INITS=5 ; Number of consecutive times to retry
0000 500 ; hardware init without waiting for awhile.
0000000A 0000 501 INIT_DELTA=10 ; Number of seconds to wait when we wait
0000 502 ; for a while.
0000 503
00000001 0000 504 ALLOC_DELTA=1 ; Number of seconds to wait before
0000 505 ; retrying allocation request.
0000 506
0000001C 0000 507 REGSAVE=2+2+4+4+16 ; Size of data saved in PU REGDUMP.
0000 508 ; Includes space for ATTNCODE(2),
0000 509 ; NUMBINITS(2), MAPREG(4),
0000 510 ; UDASA(4-MOVZWL), HOST-PORT(16).
00000001 0000 511 INIT_ATTNCODE=1 ; Attention code for INIT record.
00000002 0000 512 FAIL_ATTNCODE=2 ; Attention code for failing INIT record.
00000003 0000 513 UDASA_ATTNCODE=3 ; Attention code for record after error.
00000004 0000 514 PURGE_ATTNCODE=4 ; Attention code for Purge error record.
00000005 0000 515 UCODE_ATTNCODE=5 ; Attention code for uCODE out of rev.
0000 516
00000000 0000 517 UDA50_CNTRLTYP=0 ; UDA50 controller type code.

```

+ UDA Command Packet Layout

00000001	0000	518	LESI_CNTRLTYP=1	: LESI (RC25) controller type code.
00000005	0000	519	TU81_CNTRLTYP=5	: TU81 controller type code.
00000006	0000	520	UDA50A_CNTRLTYP=6	: UDA50A controller type code.
00000007	0000	521	RDRX_CNTRLTYP=7	: RD/RX
00000003	0000	522	MAYA_CNTRLTYP=3	: MAYA
0000000D	0000	523	QDA50_CNTRLTYP=13	: QDA50
	0000	524		
00000000	0000	525	DISK_CONID=0	: Disk Server Connection ID
00000001	0000	526	TAPE_CONID=1	: Tape Server Connection ID
00000002	0000	527	DUP_CONID=2	: DUP Server Connection ID
	0000	528		
0000000F	0000	529	SA_POLL_INTVAL=15	: SA register polling interval.
00000001	0000	530	NO_PHYCONTIGPGS=1	: # of physically contiguous pages
	0000	531		: allocate for non-aligned transfers
	0000	532		: on QBUS.
00000200	0000	533	NO_PHYCONTIGBYT=NO_PHYCONTIGPGS*512	: # of physically contiguous bytes.

+ Define PU Port specific PDT extension

```
0000 535 .SBTTL + Define PU Port specific PDT extension
0000 536
0000 537 ;+
0000 538 : PUPDT -- Define PU port specific extension to the PDT
0000 539 : (Must appear after the definition of UDASK_RINGEXP and
0000 540 : UDASK_RINGSIZE)
0000 541 :-
0000 542
0000 543 $DEFINI PUPDT
0000 544
000000E4 0000 545 .=.+PDT$C_LENGTH ; Position to end of port-
00E4 546 ; independent portion of PDT
00E4 547 $DEF PDT$C_PU_CDTARY ; Array of pointers to CDT's.
00E4 548 $DEF PDT$C_PU_VCO .BLKL 1 ; Connection block address for
00E8 549 ; virtual circuit 0 Disk MSCP
00E8 550
00E8 551 $DEF PDT$C_PU_VC1 .BLKL 1 ; Connection block address for
00EC 552 ; virtual circuit 1 Tape MSCP
00EC 553
00EC 554 $DEF PDT$C_PU_VC2 .BLKL 1 ; Connection block address for
00F0 555 ; virtual circuit 2 DUP
00F0 556
00F0 557 $DEF PDT$C_PU_VC255 .BLKL 1 ; Connection block address for
00F4 558 ; virtual circuit 255 (-1)
00F4 559 ; Maintenance protocol
00F4 560
00F4 561 $DEF PDT$C_PU_CRDARY ; Array of initial credits.
00F4 562 $DEF PDT$C_PU_CRED0 .BLKW 1 ; Credits to assign on next
00F6 563 ; CONNECT to VCO.
00F6 564 $DEF PDT$C_PU_CRED1 .BLKW 1 ; Credits to assign on next
00F8 565 ; CONNECT to VC1.
00F8 566 $DEF PDT$C_PU_CRED2 .BLKW 1 ; Credits to assign on next
00FA 567 ; CONNECT to VC2.
00FA 568 $DEF PDT$C_PU_CRD255 .BLKW 1 ; Credits to assign on next
00FC 569 ; CONNECT to VC255.
00FC 570
00FC 571 $DEF PDT$C_PU_SB .BLKL 1 ; Address of our System Block.
0100 572
0100 573 $DEF PDT$C_PU_CSR .BLKL 1 ; Pointer to controller CSR.
0104 574
0104 575 $DEF PDT$C_PU_FQPTR .BLKL 1 ; Pointer to Free Q. Used to
0108 576 ; test for emptiness of Free Q.
0108 577 $DEF PDT$C_PU_FQFL .BLKL 1 ; Free Q forward pointer.
010C 578 $DEF PDT$C_PU_FQBL .BLKL 1 ; Free Q backward pointer.
0110 579
0110 580 $DEF PDT$C_PU_SNDQFL .BLKL 1 ; Send Q forward pointer.
0114 581 $DEF PDT$C_PU_SNDQBL .BLKL 1 ; Send Q backward pointer.
0118 582
0118 583 $DEF PDT$C_PU_BUFQFL .BLKL 1 ; Buffer wait Q forward pointer.
011C 584 $DEF PDT$C_PU_BUFQBL .BLKL 1 ; Buffer wait Q backward pointer.
0120 585
0120 586 $DEF PDT$C_CRINGINX .BLKB 1 ; Index of next available slot
0121 587 ; in command ring.
0121 588 $DEF PDT$C_CPOLLINX .BLKB 1 ; Index of next slot to be polled
0122 589 ; by host, to see if released.
0122 590 $DEF PDT$C_CRINGCNT .BLKB 1 ; Count of number of command
0123 591 ; ring slots in use. Also
```



```
0123 592 ; absolute difference between
0123 593 ; previous two fields.
0123 594
0123 595 $DEF PDT$B_RRINGINX .BLKB 1 ; Index of next available slot
0124 596 ; in response ring.
0124 597 $DEF PDT$B_RPOLLINX .BLKB 1 ; Index of next slot to poll by
0125 598 ; host, to see if filled in.
0125 599 $DEF PDT$B_RRINGCNT .BLKB 1 ; Count of not yet filled in
0126 600 ; buffers on response ring.
0126 601 ; Absolute difference between
0126 602 ; previous two fields.
0126 603
0126 604 $DEF PDT$B_NOCURCON .BLKB 1 ; # of current connections.
0127 605 $DEF PDT$B_CONBITMAP .BLKB 1 ; Current Connection bit map. Corresponding
0128 606 ; bit set for each open connection.
0128 607 $DEF PDT$B_SERVERS .BLKB 1 ; Servers Supported at this Port. Bit
0129 608 ; set for each supported Server.
0129 609
0129 610 $DEF PDT$B_DATAPATH .BLKB 1 ; Permanently (or semi-permanently)
012A 611 ; allocated datapath. For VAX11-750
012A 612 ; only. Else zero.
012A 613 $DEF PDT$B_BDPUSECNT .BLKB 1 ; Count of # commands using above BDP.
0000012C 012B 614 ; Reserved
012C 615
012C 616 $DEF PDT$B_CRCONTENT .BLKB UDASK_RINGSIZE ; Array of bytes (one per each
013C 617 ; command ring slot) which
013C 618 ; maintain index of buffer
013C 619 ; currently filling this slot.
013C 620
013C 621 $DEF PDT$B_RRCONTENT .BLKB UDASK_RINGSIZE ; Array of bytes (one per each
014C 622 ; response ring slot) which
014C 623 ; maintain index of buffer
014C 624 ; currently filling this slot.
014C 625
014C 626 $DEF PDT$B_BDTABLE .BLKL 2*UDASK_RINGSIZE ; Array of longwords (one per
01CC 627 ; buffer) which point to the
01CC 628 ; respective buffers.
01CC 629
01CC 630 ; For uVAX I the communications area must not straddle a 64KB boundary.
01CC 631 ; So the following .ALIGN starts the communications area on a new page
01CC 632 ; and since it is less than 512 bytes long, it cannot straddle pages.
01CC 633 ; Note, for uVAX I the PDT is allocated in contiguous memory on a page
01CC 634 ; boundary and therefore the .ALIGN really is relevant to a page.
01CC 635
00000200 01CC 636 .=<.+511>&<^c511> ; Equivalent of .ALIGN PAGE
0200 637
00000202 0200 638 $DEF PDT$B_COMAREA ; UDA communication area base
00000203 0200 639 .BLKW 1 ; Reserved word
0202 640 .BLKB 1 ; Reserved byte
0203 641 $DEF PDT$B_PURGEDP .BLKB 1 ; Purge data path number
0204 642
0204 643 $DEF PDT$B_CMDINT .BLKW 1 ; Command ring transition flag
0206 644
0206 645 $DEF PDT$B_RSPINT .BLKW 1 ; Response ring transition flag
0208 646
0208 647 $DEF PDT$B_RINGBASE
0208 648 $DEF PDT$B_RSPRING .BLKL UDASK_RINGSIZE ; Response ring
```


+ Define PU Port specific PDT extension

```
00000088 0248 649
0248 650 $DEF PDT$ _CMDRING .BLKL UDASK_RINGSIZE ; Command ring
0288 651 PDT$ _COMAREA_LEN = .-PDT$ _COMAREA ; Length of Comarea
0288 652
0288 653 $DEF PDT$ _PU_BUFARY ; Buffer array.
0288 654
0288 655 .REPT 2*UDASK_RINGSIZE ; Allocate packets for response
0288 656 ; and command rings
0288 657
0288 658 .BLKB UDAB$ _LENGTH ; Length of a buffer.
0288 659
00000A88 0288 660 .ENDR
0C88 661 PDT$ _RINGLEN = .-PDT$ _COMAREA ; Length of area to map
0C88 662
0C88 663 $DEF PDT$ _TRTABLE .BLKL 1 ; Pointer to base of usable (beyond
0C8C 664 ; header) portion of trace table
0C8C 665 $DEF PDT$ _TRTBLPTR .BLKL 1 ; Pointer to next area in trace table
0C90 666 ; to use.
0C90 667 $DEF PDT$ _TRTBLEND .BLKL 1 ; Pointer beyond end of trace table.
0C94 668
00000C94 0C94 669
0C94 670 PDT$ _PULENGTH = . ; Total size of a
0C94 671 ; PDT for the UDA port
0C94 672
0C94 673 ; The following PDT extension is only needed when running on uVAX I.
0C94 674
0C94 675 $DEF PDT$ _PQ_MAP .BLKL 1 ; Vitrual pointer to 496 pseudo map
0C98 676 ; "registers".
0C98 677
0C98 678 $DEF PDT$ _PQ_PGQFL .BLKL 1 ; Queue Header for CDRP's waiting to
0C9C 679 $DEF PDT$ _PQ_PGQBL .BLKL 1 ; allocate the aligned Page that follows.
0CA0 680 $DEF PDT$ _PQ_POWNER .BLKL 1 ; CDRP that currently owns the Page.
0CA4 681 $DEF PDT$ _PQ_PGPHAD .BLKL 1 ; Physical address of the Page.
0CA8 682 $DEF PDT$ _PQ_UBFSVA .BLKL 1 ; Virtual address of the page mapped by
0CAC 683 ; UCB$ _SVPN. This page is used to map
0CAC 684 ; one page of the user's buffer in order
0CAC 685 ; to be able to copy it to the aligned
0CAC 686 ; Page.
0CAC 687 $DEF PDT$ _PQ_SVPSTE .BLKL 1 ; Pointer to the PTE of the UCB$ _SVPN.
0CB0 688 $DEF PDT$ _PQ_USRSTE .BLKL 1 ; Pointer into user page table that
0CB4 689 ; points to current user SVAPTE.
0CB4 690
0CB4 691 $DEF PDT$ _PQ_ALGNPG .BLKB NO_PHYCONTIGBYT ; A word aligned string of
0EB4 692 ; physically contiguous pages.
0EB4 693
00000CB4 0EB4 694 PDT$ _CONTIGLEN = .-PDT$ _COMAREA ; Length of area needed to be Physically
0EB4 695 ; contiguous.
00000EB4 0EB4 696 PDT$ _UV1LENGTH = . ; Length of PDT needed for uVAX I.
0EB4 697 $DEFEND PUPDT
```

+ Define PU specific UCB extension

```

0000 699          .SBTTL +          Define PU specific UCB extension
0000 700
0000 701      ;+
0000 702      ; $PUUCBDEF -- Defined UDA extension to UCB.
0000 703      ; (Must appear after the definition of UDASK_RINGEXP and
0000 704      ; UDASK_RINGSIZE)
0000 705      ; -
0000 706
0000 707          $DEFINI UCB
0000 708
000000A0 0000 709      . =UCBSL_DPC+4          ; Position to end of
00A0 710          ; standard UCB for
00A0 711          ; error logging devices
00A0 712
00A0 713 $DEF      UCBSL_PU_ALLOC .BLKL 1          ; Space to save size of alloc. request.
00A4 714
00A4 715 $DEF      UCBSB_UDAFLAGS .BLKB 1          ; Internal control flags
00A5 716      $VIELD      UDA,0,<-          ; Internal flag definitions
00A5 717          <ONLINE,,V>,-          ; UDA is On Line
00A5 718          <STOPPED,,M>,-          ; Port only open for DUP CONNECTIONS.
00A5 719      >
00A5 720
00A5 721 $DEF      UCBSB_INITCNT .BLKB 1          ; Count of # of times left to retry
00A6 722          ; hardware init consecutively.
00A6 723 $DEF      UCBSW_NUMBINITS .BLKW 1          ; Number of times UDA50 has been Initied.
00A8 724
00A8 725 $DEF      UCBSW_ATTNCODE .BLKW 1          ; Attention code for PU_REGDUMP.
00AA 726
00AA 727 $DEF      UCBSW_UDASA .BLKW 1          ; Contents of SA saved at last interrupt
00AC 728
00AC 729 $DEF      UCBSW_PORTSTEP1 .BLKW 1          ; Contents of SA at start of STEP1.
00AE 730      _VIELD      PS1,0,<-          ; Fields in STEP1
00AE 731          <,6>,-          ; Reserved
00AE 732          <MP,,M>,-          ; If set port supports address mapping
00AE 733          <OD,,M>,-          ; If set port supports odd addresses
00AE 734          <DI,,M>,-          ; If set port implements enhanced diagnosis
00AE 735          <QB,,M>,-          ; If set port supports 22-bit bus addr
00AE 736          <NV,,M>,-          ; If set no host settable vector addr
00AE 737          <S1,,M>,-          ; Must be set in step 1
00AE 738          <S2,,M>,-          ; Must be clear in step 1
00AE 739          <S3,,M>,-          ; Must be clear in step 1
00AE 740          <S4,,M>,-          ; Must be clear in step 1
00AE 741          <ER,,M>,-          ; If set then we had an error
00AE 742      >
00AE 743
00AE 744 $DEF      UCBSW_HOSTSTEP1 .BLKW 1          ; What host writes to SA in STEP1.
00B0 745      _VIELD      HS1,0,<-          ; Fields in Host STEP1
00B0 746          <INTVEC,7,M>,-          ; Interrupt vector address/4
00B0 747          <IE,,M>,-          ; Interrupt enable during STEPS 1-3
00B0 748          <RRNGLEN,3,M>,-          ; Response ring length (exponent)
00B0 749          <CRNGLEN,3,M>,-          ; Command ring length (exponent)
00B0 750          <WR,,M>,-          ; If set port enters diagnostic wrap mode
00B0 751          <BIT15,,M>,-          ; High bit always set
00B0 752      >
00B0 753
00B0 754 $DEF      UCBSW_PORTSTEP2 .BLKW 1          ; What port responds at start of STEP2.
00B2 755      _VIELD      PS2,0,<-          ; Fields in Port STEP2

```


+ Define PU specific UCB extension

```

00B2 756 <RRNGLEN,3,M>,- ; Echoed response ring length (exponent)
00B2 757 <CRNGLEN,3,M>,- ; Echoed command ring length (exponent)
00B2 758 <WR,,M>,- ; Echoed diagnostic wrap bit
00B2 759 <1>,- ; Echoed BIT15 always set
00B2 760 <PORTTYPE,3,M>,- ; Type of this port
00B2 761 <S1,,M>,- ; Must be clear in STEP2
00B2 762 <S2,,M>,- ; Must be set in STEP2
00B2 763 <S3,,M>,- ; Must be clear in STEP2
00B2 764 <S4,,M>,- ; Must be clear in STEP2
00B2 765 <ER,,M>,- ; If set then error in STEP1
00B2 766 >
00B2 767
00B2 768 $DEF UCBSW_HOSTSTEP2 .BLKW 1 ; What host writes to SA during STEP2.
00B4 769 _VIELD HS2,0,<- ; Fields in Host STEP2
00B4 770 <PI,,M>,- ; Host requests adapter purge interrupts
00B4 771 <RINGBASEL,15,M>,- ; Lo order of address of communication
00B4 772 > ; area (in UNIBUS virtual space)
00B4 773
00B4 774 $DEF UCBSW_PORTSTEP3 .BLKW 1 ; What port responds at start of STEP3.
00B6 775 _VIELD PS3,0,<- ; Fields in Port STEP3
00B6 776 <INTVEC,7,M>,- ; Echoed interrupt address/4
00B6 777 <IE,,M>,- ; Echoed interrupt enable
00B6 778 <3>,- ; Reserved
00B6 779 <S1,,M>,- ; Must be clear in STEP3
00B6 780 <S2,,M>,- ; Must be clear in STEP3
00B6 781 <S3,,M>,- ; Must be set in STEP3
00B6 782 <S4,,M>,- ; Must be clear in STEP3
00B6 783 <ER,,M>,- ; If set we had error in STEP2
00B6 784 >
00B6 785
00B6 786 $DEF UCBSW_HOSTSTEP3 .BLKW 1 ; What host writes to SA during STEP3.
00B8 787 _VIELD HS3,0,<- ; Fields in Host STEP3
00B8 788 <RINGBASEH,15,M>,- ; Hi order of address of comm. area
00B8 789 <PP,,M>,- ; If set host requests execution of
00B8 790 > ; purge and poll tests
00B8 791
00B8 792 $DEF UCBSW_PORTSTEP4 .BLKW 1 ; What port responds at start of STEP4.
00BA 793 _VIELD PS4,0,<- ; Fields in Port STEP4
00BA 794 <UCODEVER,4,M>,- ; Microcode version number
00BA 795 <CNTRLTYP,7,M>,- ; Controller type
00BA 796 <S1,,M>,- ; Must be clear in STEP4
00BA 797 <S2,,M>,- ; Must be clear in STEP4
00BA 798 <S3,,M>,- ; Must be clear in STEP4
00BA 799 <S4,,M>,- ; Must be set in STEP4
00BA 800 <ER,,M>,- ; If set we had an error in STEP3
00BA 801 >
00BA 802
00BA 803 $DEF UCBSW_HOSTSTEP4 .BLKW 1 ; What host writes to SA during STEP4.
00BC 804 _VIELD HS4,0,<- ; Fields in Host STEP4
00BC 805 <GO,,M>,- ; If set controller begins immediately
00BC 806 <LF,,M>,- ; If set host wants Last Fail response
00BC 807 <BURST,6,M>,- ; Maximum lonwords(-1) / NPR transfer
00BC 808 <.8>,- ; Reserved
00BC 809 >
00BC 810
00BC 811
00BC 812 $DEF UCBSL_PU_SVAPTE .BLKL 1 ; Place to save UCBSL_SVAPTE.

```

+ Define PU specific UCB extension

```

00C0 813 $DEF UCB$W_PU_BOFF .BLKW 1 ; Place to save UCB$W_BOFF.
00C2 814 $DEF UCB$W_PU_BCNT .BLKW 1 ; Place to save UCB$W_BCNT.
00C4 815
00C4 816 $VFIELD UCB,0,<- ; Define bits for UCB$W_DEVSTS
00C4 817 <,1>,- ; Unused
00C4 818 <PU_FRKBSY,,M>,- ; Fork block interlock bit
00C4 819 <PU_HRDINI,,M>,- ; Hardware init in progress
00C4 820 <PU_PWINIT,,M>,- ; POWER init in progress
00C4 821 <PU_MRESET,,M>,- ; Goto POST_POWER_FORK after fork.
00C4 822 <PU_BDPATH,,M>,- ; BDP already allocated (or not).
00C4 823 <PU_INILOG,,M>,- ; Init Log in progress
00C4 824 >
00C4 825
000000C4 00C4 826 UCB$C_PUSIZE = .
00C4 827
00C4 828 $DEFEND UCB

```


DRIVER STRUCTURES

H 9

16-SEP-1984 01:05:05 VAX/VMS Macro V04-00
5-SEP-1984 00:17:10 [DRIVER.SRC]PUDRIVER.MAR;1

Page 18
(2)

```

0000 830 .SBTTL DRIVER STRUCTURES
0000 831 .SBTTL + Driver Prologue Table
0000 832
0000 833 DPTAB END=PU$END,- ;End of driver label
0000 834 ADAPTER=UBA,- ;Adapter type
0000 835 FLAGS=DPT$M_SCS!DPT$M_SVP!DPT$M_NOUNLOAD,- ;Driver requires SCS to be loaded
0000 836 - ; uVAX I needs System Virtual Page
0000 837 -
0000 838 UCBSIZE=UCB$C_PUSIZE,- ;UCB size
0000 839 NAME=PUDRIVER ;Driver name
0038 840
0038 841 DPT_STORE INIT
0038 842
0038 843 DPT_STORE UCB,UCB$B_FIPL,B,8 ;Fork IPL
003C 844
003C 845 DPT_STORE UCB,UCB$L_DEVCHAR,L,<- ;Device characteristics:
003C 846 DEV$M_SHRT,- ; Sharable
003C 847 DEV$M_AVL!- ; Available
003C 848 DEV$M_ELG!- ; Error logging device
003C 849 DEV$M_IDV!- ; Input device
003C 850 DEV$M_ODV> ; Output device
0043 851
0043 852 DPT_STORE UCB,UCB$B_DIPL,B,21 ;Device interrupt IPL
0047 853
0047 854 DPT_STORE REINIT
0047 855
0047 856 DPT_STORE DDB,DDB$L_DDT,D,PU$DDT ;DDT address
004C 857 DPT_STORE CRB,CRB$L_INTD+4,- ;Interrupt routine addr
004C 858 D,PU$INT
0051 859 DPT_STORE CRB,CRB$L_INTD+VEC$L_INITIAL,-
0051 860 D,PU$CTINIT ;Controller init addr
0056 861 DPT_STORE CRB,CRB$L_INTD+VEC$L_UNITINIT,-
0056 862 D,PU$UNITINIT ;Unit init addr
005B 863 DPT_STORE END

```

+ Driver Dispatch Table

.SBTTL + Driver Dispatch Table

DDTAB DEVNAM=PU,-
START=PU\$STARTIO,-
FUNCTB=PU\$FUNCTABLE,-
UNITINIT=PU\$UNITINIT,-
ERLGBF=REGSAVE+4+EMB\$\$_DV_REGSAV,-
REGDMP=PU_REGDUMP

; QIO's are illegal temporarily
; Function decision table
; Unit init routine addr

We only support two functions: IOS_STOP and IOS_INITIALIZE. The first effectively closes the Port to all connections except those to the DUP SERVER. The second function reopens the Port and initializes it.

Inputs:
R3 => IRP
R5 => UCB of Port

PU\$STARTIO:

		00	EF	0038	887	EXTZV	#IRPSV_FCODE,-	; Extract function code.
		06		003A	888		#IRPSS_FCODE,-	
50	20	A3		003B	889		IRPSW_FUNC(R3),R0	
	50	03	91	003E	890	CMPB	#IOS_STOP,R0	; See if IOS_STOP request.
		20	13	0041	891	BEQL	START_STOP	; EQL means yes it was.
	50	04	91	0043	892	CMPB	#IOS_INITIALIZE,R0	; See if IOS_INITIALIZE requested.
		38	12	0046	893	BNEQ	ILLIOFUNC	; NEQ means not one of supported functions.
		55	DD	0048	894			
		2F	10	004A	895	PUSHL	R5	; Save R5 => UCB.
				004C	896	BSBB	SUCCESS	; Complete QIO request but get control
		55	8ED0	004C	897			; after REQCOM so as to continue.
				004F	898	POPL	R5	; Restore R5 => UCB.
54	0084	C5	D0	0055	899	REQPCHAN		; Re-allocate channel after REQCOM.
		OCA2	30	005A	900	MOVL	UCB\$_PDT(R5),R4	; R4 => PDT.
		02	8A	005D	901	BSBW	FPC\$MRESET	; Do initialize.
	00A4	C5		005F	902	BICB	#UDAS\$ STOPPED,-	; Open up port.
			05	0062	903		UCB\$_ODAFLAGS(R5)	
				0063	904	RSB		; Kill this thread.
		55	DD	0063	905	START_STOP:		
		14	10	0065	906	PUSHL	R5	; Save R5 => UCB.
				0067	907	BSBB	SUCCESS	; Complete QIO request but get control
		55	8ED0	0067	908			; after REQCOM so as to continue.
				006A	909	POPL	R5	; Restore R5 => UCB.
54	0084	C5	D0	0070	910	REQPCHAN		; Re-allocate channel after REQCOM.
		02	88	0075	911	MOVL	UCB\$_PDT(R5),R4	; R4 => PDT.
	00A4	C5		0077	912	BISB	#UDAS\$ STOPPED,-	; Close port.
			05	007A	913		UCB\$_ODAFLAGS(R5)	
				007B	914	RSB		; Kill this thread.
	50	01	D0	007B	915	SUCCESS:		
		05	11	007E	916	MOVL	#SS\$ NORMAL,R0	; Return success code.
				0080	917	BRB	COMPLETE_IO	; And branch around.
50	00F4	8F	3C	0080	918	ILLIOFUNC:		
				0085	919	MOVZWL	#SS\$_ILLIOFUNC,R0	; We do not support other QIO's
		51	D4	0085	920	COMPLETE_IO:		
				0085	921	CLRL	R1	

PUDRIVER
V04-000

+ Driver Dispatch Table

0087 922

REQCOM

J 9

16-SEP-1984 01:05:05
5-SEP-1984 00:17:10

VAX/VMS Macro V04-00
[DRIVER.SRC]PUDRIVER.MAR;1

Page 20
(2)

+ "Register" Dump routine

```

008D 924      .SBTTL +      "Register" Dump routine
008D 925
008D 926      + PU_REGDUMP
008D 927      :
008D 928      Inputs:
008D 929          R0 => Buffer to fill
008D 930          R5 => UCB
008D 931      :
008D 932      Outputs:
008D 933          R1 modified
008D 934      :
008D 935
008D 936      PU_REGDUMP:
008D 937
80    80    1C    D0 008D 938      MOVL      #REGSAVE,(R0)+      ; Save number of bytes following.
00A8 C5    B0 0090 939      MOVW      UCB$W_ATTNCODE(R5),(R0)+; Save Attention code (record type).
00A6 C5    B0 0095 940      MOVW      UCB$W_NUMBINIT(S(R5),(R0)+; Number times port init'ed.
51    24 A5    D0 009A 941      MOVL      UCB$W_CRB(R5),R1      ; R1 => CRB.
009E 942
009E 943      ASSUME      VEC$B_NUMREG      EQ      VEC$W_MAPREG+2
009E 944      ASSUME      VEC$B_DATAPATH    EQ      VEC$B_NUMREG+1
80    34 A1    D0 009E 945      MOVL      CRB$W_INTD+VEC$W_MAPREG(R1),(R0)+      ; Save dedicated map registe
00A2 946
80    00AA C5    3C 00A2 947      MOVZWL   UCB$W_UDASA(R5),(R0)+      ; Save contents of SA register.
00A7 948
00A7 949      ASSUME      UCB$W_HOSTSTEP1 EQ      UCB$W_PORTSTEP1+2
00A7 950      ASSUME      UCB$W_PORTSTEP2 EQ      UCB$W_HOSTSTEP1+2
00A7 951      ASSUME      UCB$W_HOSTSTEP2 EQ      UCB$W_PORTSTEP2+2
80    00AC C5    7D 00A7 952      MOVQ      UCB$W_PORTSTEP1(R5),(R0)+      ; Save Host-Port dialogue.
00AC 953
00AC 954      ASSUME      UCB$W_HOSTSTEP3 EQ      UCB$W_PORTSTEP3+2
00AC 955      ASSUME      UCB$W_PORTSTEP4 EQ      UCB$W_HOSTSTEP3+2
00AC 956      ASSUME      UCB$W_HOSTSTEP4 EQ      UCB$W_PORTSTEP4+2
80    00B4 C5    7D 00AC 957      MOVQ      UCB$W_PORTSTEP3(R5),(R0)+      ; Save Host-Port dialogue.
00B1 958
05    00B1 959      RSB      ; Return to caller.

```


+ Function Decision Table

00B2 961
00B2 962
00B2 963
00B2 964
00B2 965
00B2 966
00B2 967
00BA 968
00BA 969
00BA 970
00BA 971
00C2 972
00C2 973
00C2 974

.SBTTL +

Function Decision Table

PUSFUNCTIONTABLE:

FUNCTAB

;-
<STOP,-
INITIALIZE>

;Valid functions:
; Stop port.
; Reopen Port and Initialize.

FUNCTAB

;-
<STOP,-
INITIALIZE>

;Buffered functions:
; Stop port.
; Reopen Port and Initialize.

FUNCTAB

+EXE\$ZEROPARM,-
<STOP,-
INITIALIZE>

;ZERO PARAMETER FUNCTIONS
; Stop port.
; Reopen Port and Initialize.

+ Static Storage

	00CE	976	.SBTTL +	Static Storage	
	00CE	977			
	00CE	978	; Static driver storage		
	00CE	979			
00'00'	00CE	980	.ALIGN LONG,0		
00000000	00D0	981	PUSL DRIVER_STS:	.LONG 0	; Status flag longword.
00000000	00D4	982	MAP\$V_MAPREGS=0		; Bit meaning that the pseudo map
	00D4	983			; registers have been mapped into
	00D4	984			; system space at ^x800 in the Q-bus
	00D4	985			; "CSR" area pointed at by ADP.
00000001	00D4	986	MAP\$M_MAPREGS=1		; Mask of above bit.
00000000	00D4	987	PUSL_MPHYAD:	.LONG 0	; Physical address of "map registers".
00000000	00D8	988	PUSL_TRACE_VARIABLE:	.LONG 0	; Variable describing what kinds (if
	00DC	989			; any) of trace have been enabled.
	00DC	990			; 0 implies no port tracing.
	00DC	991			; 1 implies we try to allocate resources
	00DC	992			; to trace for ports that support Tapes
	00DC	993			; 2 implies that we try to allocate
	00DC	994			; resources to trace all ports

+ NULL MESSAGE AND DATAGRAM INPUT ROUTIN

```

00DC 996      .SBTTL +      NULL MESSAGE AND DATAGRAM INPUT ROUTINES
00DC 997      .SBTTL +      NULL CDT
00DC 998
00DC 999      ;+
00DC 1000     ; NULL CDT and NULL message input and datagram input routines.
00DC 1001     ;
00DC 1002     ; Inputs:                (for routines)
00DC 1003     ;
00DC 1004     ; R1                    -Length of message
00DC 1005     ; R2                    -Addr of message
00DC 1006     ; R4                    -Addr of PDT
00DC 1007     ; -
00DC 1008
00DC 1009     .enabl lsb
00DC 1010
00DC 1011     NULL_MSG_INPUT:          ; Sequential messages for null connections
00DC 1012     ; are to be ignored.
53 DD 00DC 1013     PUSHL R3           ; Save R3.
10 11 00DE 1014     BRB 10$           ; Branch around to deallocate buffer.
00E0 1015
00E0 1016     NULL_DG_INPUT:          ; Datagrams for null connections are to
00E0 1017     ; be logged.
53 DD 00E0 1018     PUSHL R3           ; Save R3.
53 DD 00E2 1019     MOVL PDT$LC_UCB0(R4),R3 ; R3 => UCB. (input to ERL$LOGMESSAGE)
50 04 DD 00E7 1020     MOVL #EMB$C UM,R0 ; Log message type.
00000000'GF 16 00EA 1021     JSB G*ERL$LOGMESSAGE ; Call to log message.
00F0 1022     10$:
52 14 C2 00F0 1023     SUBL #UDAB$T TEXT,R2 ; R2 => buffer header.
0DA7 30 00F3 1024     BSBW Q_DEALLOC_BUF ; Call internal entry to deallocate buffer.
53 8ED0 00F6 1025     POPL R3          ; Restore R3.
00F9 1026     NULL_ROUTINE:          ; Label of an RSB instruction.
00F9 1027     NULL_ERR_ROUT:
05 00F9 1028     RSB
00FA 1029
00'00' 00FA 1030     .ALIGN LONG,0
00FC 1031     NULL_CDT:
0000019C 00FC 1032     .BLKB CDT$K_LENGTH ; Allocate space for the NULL CDT.
019C 1033     .dsabl lsb

```

Request and Release DATAPATH transfer ve

```
019C 1035 .SBTTL Request and Release DATAPATH transfer vectors
019C 1036
019C 1037 ;+
019C 1038
019C 1039 .ALIGN LONG,0
019C 1040
019C 1041 REQDATAPATH_TV: ; Transfer vector to routine to request
019C 1042 ; a UNIBUS datapath.
00000000' 019C 1043 .LONG IOC$REQDATAPUDA ; Routine to call (for VAX-11/780).
01A0 1044 ; For other processors, this pointer
01A0 1045 ; will be overlaid with the address of
01A0 1046 ; REQDATAP_750 or REQDATAP_730, below.
01A0 1047
01A0 1048 RELDATAPATH_TV: ; Transfer vector to routine to release
01A0 1049 ; a UNIBUS datapath.
00000000' 01A0 1050 .LONG IOC$RELDATAPUDA ; Routine to call when running on
01A4 1051 ; VAX-11/780. For other processors,
01A4 1052 ; this address is overlaid with the
01A4 1053 ; RELDATAP_750 or RELDATAP_730, below.
01A4 1054
01A4 1055 ;
01A4 1056 ; REQDATAP_750 assigns the "semi-permanently" allocated datapath, currently
01A4 1057 ; in use by the current burst, bumping the useage count by one,
01A4 1058 ; or if no burst is in progress, allocates a buffered datapath and
01A4 1059 ; then increments the useage count by one.
01A4 1060
01A4 1061 ; The following paragraph applies only until the Buffered Datapath bug is
01A4 1062 ; fixed on VAX-11/750's.
01A4 1063 ; If the current transfer is NOT longword aligned or if it is NOT
01A4 1064 ; an integral number of longwords in length, we do NOT use the Buffered
01A4 1065 ; Datapath but instead use the direct datapath. We implicitly assign
01A4 1066 ; direct datapath (i.e. datapath zero) by simply
01A4 1067 ; RSB'ing to our caller, depending on the fact that our caller cleared
01A4 1068 ; CDRP$L_UBARSRC before calling us. Note that in this case we do NOT
01A4 1069 ; increment any useage count.
01A4 1070
01A4 1071 ; REQDATAP_730 implicitly assigns the direct datapath to the current transfer
01A4 1072 ; since the 730 has no buffered datapaths (i.e. datapath zero). This is
01A4 1073 ; accomplished by simply RSB'ing to our caller, depending on the fact
01A4 1074 ; that our caller cleared CDRP$L_UBARSRC before calling us.
01A4 1075
01A4 1076
01A4 1077 ; Inputs:
01A4 1078 ; R4 -Addr of PDT
01A4 1079 ; R5 -Addr of CDRP
01A4 1080
01A4 1081 ; Outputs: (for REQDATAP_750 and REQDATAP_730)
01A4 1082 ; CDRP$L_UBARSRC+UBMDSB_DATAPATH set from PDT$B_DATAPATH.
01A4 1083 ;
01A4 1084
01A4 1085 REQDATAP_750:
01A4 1086
01A4 1087 BISB3 CDRP$L_BCNT(R5),- ; R0 has "OR" of low
01A7 1088 CDRP$W_BOFF(R5),R0 ; bits of BOFF and BCNT,
01AA 1089 BITB #3,R0 ; Test for longword alignment
01AD 1090 ; and integral # longwords.
50 D2 A5 89 01AD 1091 BNEQ REQDATAP_730 ; NEQ means not aligned or
50 D0 A5 93
50 03 12
34 12
```



```
01AF 1092 ; possibly odd # of words
01AF 1093 ; being transferred.
01AF 1094 ; Else fall thru to use pre-
01AF 1095 ; allocated datapath.
01AF 1096 REQDATAP_8SS: ; UQport on a BUA.
012A C4 95 01AF 1097 TSTB PDT$B_BDPUSECNT(R4) ; Do we have semi-perm BDP?
OC 12 01B3 1098 BNEQ 10$ ; NEQ implies yes. Goto use it.
00000000'GF 16 01B5 1099 JSB G^IOC$REQDATAPUDA ; Call to allocate a BDP.
3F A5 90 01BB 1100 MOVB CDRP$L_UBARSRC+UBMD$B_DATAPATH(R5),- ; Record new semi-perm
0129 C4 01BE 1101 PDT$B_DATAPATH(R4) ; BDP in PDT.
01C1 1102 10$:
0129 C4 90 01C1 1103 MOVB PDT$B_DATAPATH(R4),- ; Assign semi-perm data-
3F A5 01C5 1104 CDRP$L_UBARSRC+UBMD$B_DATAPATH(R5) ; path to transfer.
012A C4 96 01C7 1105 INCB PDT$B_BDPUSECNT(R4) ; Inc usage count.
05 01CB 1106 RSB ; Return to caller.
01CC 1107
01CC 1108 RELDATAP_750:
01CC 1109
50 D2 A5 89 01CC 1110 BISB3 CDRP$L_BCNT(R5),- ; R0 has 'OR' of low
D0 A5 01CF 1111 CDRP$W_BOFF(R5),R0 ; bits of BOFF and BCNT,
50 03 93 01D2 1112 BITB #3,R0 ; Test for longword alignment
OC 12 01D5 1113 BNEQ REQDATAP_730 ; and integral # longwords.
01D7 1114 ; NEQ means not aligned or
01D7 1115 ; possibly odd # of words
01D7 1116 ; being transferred.
01D7 1117 ; Else fall thru to use pre-
01D7 1118 ; allocated datapath.
012A C4 97 01D7 1119 RELDATAP_8SS: ; UQport on a BUA.
06 12 01DB 1120 DECB PDT$B_BDPUSECNT(R4) ; DECR usage count.
00000000'GF 16 01DD 1121 BNEQ REQDATAP_730 ; NEQ means not end of burst.
01E3 1122 JSB G^IOC$RE[DATAPUDA ; Release BDP after burst.
01E3 1123 ;
01E3 1124 REQDATAP_730: ; Return to caller implicitly assigning
01E3 1125 ; direct datapath (i.e. datapath zero)
01E3 1126 RELDATAP_730: ; Return to caller implicitly de-assigning
01E3 1127 ; direct datapath (i.e. datapath zero)
05 01E3 1128 RSB ;
01E3 1129 ; Return to caller.
```

INITIALIZATION

```

01E4 1131      .SBTTL  INITIALIZATION
01E4 1132      ;+
01E4 1133      ; The following table gives word offsets for fork process SCS calls.
01E4 1134      ; Offsets are relative to the address of the controller initialization
01E4 1135      ; routine, PUSCTLINIT.
01E4 1136      ; -
01E4 1137
01E4 1138      ;
01E4 1139      ; Macro to generate the table and ASSUME statements about PDT format:
01E4 1140      ;
01E4 1141
01E4 1142      .MACRO  SCS_OFFSET_TAB  ENTRY_LIST
01E4 1143
01E4 1144      $$$ENTRYNUM=0                      ; No entries in table yet
01E4 1145      .IRP   ENTRY_LIST              ; For each entry in the list...
01E4 1146      .WORD  <FPCS'ENTRY'-PUSCTLINIT> ;   insert offset from ctl init,
01E4 1147      .IF    NE $$$ENTRYNUM          ;   and for entries after the 1st
01E4 1148      ASSUME $$$PREV+4 EQ PDT$$_'ENTRY' ;   specify assumed PDT adjacency
01E4 1149      .ENDC
01E4 1150      $$$PREV=PDT$$_'ENTRY'           ; Set previous entry as this entry
01E4 1151      $$$ENTRYNUM=$$$ENTRYNUM+1       ; Step entry count
01E4 1152      .ENDR
01E4 1153
01E4 1154      ASSUME $$$PREV+4 EQ PDT$_SCSEND      ; Final PDT assumption
01E4 1155      .WORD  0                          ; Offset table terminator
01E4 1156
01E4 1157      .ENDM   SCS_OFFSET_TAB              ;
01E4 1158
01E4 1159      ; Table itself:
01E4 1160      ;
01E4 1161      ;
01E4 1162      PUS$SCSOFFSET::
01E4 1163
01E4 1164      SCS_OFFSET_TAB <-
01E4 1165      ACCEPT,-                               ; Invoke macro to define offsets
01E4 1166      ALLOC$G,-
01E4 1167      ALLOCMSG,-
01E4 1168      CONNECT,-
01E4 1169      DEALLOC$G,-
01E4 1170      DEALLOMSG,-
01E4 1171      DEALRGMSG,-
01E4 1172      DCONNECT,-
01E4 1173      MAP,-
01E4 1174      MAPBYPASS,-
01E4 1175      MAPIRP,-
01E4 1176      MAPIRPBYP,-
01E4 1177      QUEUED$G,-
01E4 1178      QUEUEMD$G,-
01E4 1179      RCHMSG$BUF,-
01E4 1180      RCLMSG$BUF,-
01E4 1181      REJECT,-
01E4 1182      REQDATA,-
01E4 1183      SENDDATA,-
01E4 1184      SENDDG,-
01E4 1185      SENDMSG,-
01E4 1186      SDCNTMSG,-
01E4 1187

```


INITIALIZATION

```

01E4 1188 UNMAP,-
01E4 1189 READCOUNT,-
01E4 1190 RLSCOUNT,-
01E4 1191 MRESET,-
01E4 1192 MSTART,-
01E4 1193 MAINTFCN,-
01E4 1194 SENDRGDG,-
01E4 1195 STOP_VCS,-
01E4 1196 >
0222 1197
0222 1198 .SBTTL + UNIT_INIT
0222 1199
0222 1200 :+
0222 1201 : TBS
0222 1202 :
0222 1203 : Inputs:
0222 1204 :
0222 1205 : R3 -Address of PA configuration reg
0222 1206 : R4 -Same as R3
0222 1207 : R5 -Addr of UCB
0222 1208 :
0222 1209 : Outputs:
0222 1210 :
0222 1211 : R0 -Status code
0222 1212 : (SS$_NORMAL,SS$_INSFMEM, or 0 if no SVPN)
0222 1213 : R1-R4 -Destroyed
0222 1214 : Other registers -Preserved
0222 1215 :-
0222 1216 :
0222 1217 : .ENABL LSB
0222 1218 :
0222 1219 PUSUNITINIT::
0222 1220 :
54 A5 B5 0222 1221 : MOVB #DTS_UDA50,UCB$B_DEVTYPE ; Set the device type to UDA50
06 13 0222 1222 : TSTW UCB$W_UNIT(R5) ; Is this unit 0?
0225 1223 : BEQL INIT_CTLR ; If YES, go to init controller first.
0227 1224 :
10 A8 0227 1225 : BISW #UCB$M_ONLINE,- ; Else merely set unit on line
64 A5 0229 1226 : UCB$W_STS(R5) ; and
022B 1227 : RSB ; Return
022C 1228 : .DSABL LSB

```

+ CONTROLLER INIT

```

022C 1230 .SBTTL + CONTROLLER INIT
022C 1231
022C 1232 :+
022C 1233 : The controller initialization entry as seen by the system, PUSCTLINIT,
022C 1234 : is a noop since initialization can't begin without the unit 0 UCB.
022C 1235 : Actual controller init is called from unit 0 unit initialization with
022C 1236 : the same inputs as unit init.
022C 1237 :
022C 1238 :
022C 1239 : Inputs:
022C 1240 :
022C 1241 : R3 -Addr of PU CSR
022C 1242 : R4 -Same as R3
022C 1243 : R5 -Addr of UCB for unit 0
022C 1244 :
022C 1245 : Outputs:
022C 1246 :
022C 1247 : R0-R3 -Destroyed
022C 1248 : Other registers -Preserved
022C 1249 :-
022C 1250 :
022C 1251 PUSCTLINIT:: ; Controller init called by system
022C 1252
05 022C 1253 RSB ; Return

```


+ CONTROLLER INIT

```
022D 1255 :  
022D 1256 : Controller initialization called from unit 0 init.  
022D 1257 :  
022D 1258 : Inputs:  
022D 1259 : R5 -UCB address  
022D 1260 :  
022D 1261 : Outputs:  
022D 1262 : R0 -Status code  
022D 1263 : (SS$_NORMAL,SS$_INSFMEM, or 0 if no SVPN)  
022D 1264 :  
00000031 022D 1265 BRW_OPCODE=^x31 ; Opcode of BRW instruction, used in  
022D 1266 : concocting PATCH below.  
022D 1267 :  
022D 1268 .enabl lsb  
022D 1269 :  
022D 1270 INIT_CTLR::  
022D 1271 :  
00000000'06 11 022D 1272 BRB 0$ ; Branch around breakpoint.  
16 022F 1273 JSB G^INISBRK ; Breakpoint for debugging.  
0235 1274 0$:  
0084 C5 D5 0235 1275 TSTL UCB$_PDT(R5) ; See if first time thru here.  
7A 12 0239 1276 BNEQ POWER_INIT ; NEQ implies this is POWER FAILURE.  
FE99 CF D5 023B 1277 TSTL PUSL_TRACE_VARIABLE ; See if any tracing requested.  
12 13 023F 1278 BEQL 5$ ; EQL implies NO tracing.  
0241 1279 :  
0241 1280 : Here if tracing has been enabled. There are two places in the inline code  
0241 1281 : where we would like to trace; once in Send Sequence message, and once when  
0241 1282 : receiving a message. In order to accomplish this, we here dynamically  
0241 1283 : patch a BRW instruction into the two appropriate locations.  
0241 1284 :  
0F80'CF 00001E31'8F D0 0241 1285 MOVL #BRW_OPCODE!- ; If tracing enabled, patch code  
0242 1286 <ENABLE_COMMAND_OFFSET@8>,- ; into Send Message Logic to trace  
0242 1287 W^ENABLE_COMMAND_START ; outgoing message.  
D0 024A 1288 MOVL #BRW_OPCODE!- ; Here patch code into Receive Logic  
024B 1289 <ENABLE_RESPONSE_OFFSET@8>,- ; to trace received message.  
12BD'CF 00009031'8F 024B 1290 W^ENABLE_RESPONSE_START  
0253 1291 5$:  
02 0253 1292 BISW #UCB$_PU_FRKBSY,- ; Indicate the UCB fork block is in  
68 A5 A8 0255 1293 UCB$_DEVSTS(R5) ; use. Setting this bit defers processing  
0257 1294 ; powerfailure recovery until after  
0257 1295 ; initialization completes.  
0257 1296 IOFORK ; Lower IPL.  
025D 1297 :  
025D 1298 REQPCAN ; Permanently allocate channel.  
0263 1299 :  
80 8F 90 0263 1300 MOVB #DC$_BUS,- ; Initialize constant UCB fields.  
40 A5 0266 1301 UCB$_DEVCLASS(R5)  
03 90 0268 1302 MOVB #DT$_UQPORT,- ; Type of UQPORT will be determined  
41 A5 026A 1303 UCB$_DEVTYPE(R5) ; later.  
026C 1304 :  
026C 1305 ; Initialize CRB wakeup mechanism.  
026C 1306 :  
53 24 A5 D0 026C 1307 MOVL UCB$_CRB(R5),R3 ; R3 => CRB.  
10 A3 55 D0 0270 1308 MOVL R5,CRB$_AUXSTRUC(R3) ; CRB => UCB as auxillary structure.  
18 A3 01 CE 0274 1309 MNEGL #1,CRB$_DUETIME(R3) ; Set infinite time.  
FE7D CF 9E 0278 1310 MOVAB NULL_ROUTINE,-  
1C A3 027C 1311 CRB$_TOUTROUT(R3) ; Inocuous routine for now.
```

+ CONTROLLER INIT

```

00000000'GF 16 027E 1312 JSB G^IOC$THREADCRB ; Put CRB on wakeup list.
               0284 1313
007C 30 0284 1314 BSBW BUILD_PDT ; Allocate and fill in PDT.
2A 50 E9 0287 1315 BLBC RO,15$ ; Return error status to caller.
0258 30 028A 1316 BSBW INIT_UDA_BUFFERS ; Map area in PDT containing buffers.
24 50 E9 028D 1317 BLBC RO,15$ ; Return error status to caller.
046E 30 0290 1318 BSBW INIT_INIT_STEPS ; Initialize values in UCB to use
               0293 1319 ; in UDA hardware initialization.
1E 50 E9 0293 1320 BLBC RO,15$ ; Return error status to caller.
0926 30 0296 1321 BSBW BUILD_PB_SB ; Create and fill in system block and
               0299 1322 ; path block.
18 50 E9 0299 1323 BLBC RO,15$ ; Return error status to caller.
               029C 1324 10$:
05 90 029C 1325 MOVB #NO_CONSEC_INITS,- ; Initialize UCB field that counts
00A5 C5 029E 1326 UCB$B_INITCNT(R5) ; hardware init retries.
               02A1 1327
056D 30 02A1 1328 BSBW HARDWARE_INIT ; Init UDA hardware.
0D 50 E9 02A4 1329 BLBC RO,15$ ; LBC implies couldn't Init hardware.
09C1 30 02A7 1330 BSBW UPDATE_PB_SB ; Update fields in the Path and System block
10 A8 02AA 1331 B1SW #UCB$M_ONLINE,- ; After hardware init, we are online.
64 A5 02AC 1332 BSBW UCB$W_STS(R5)
0FAC 30 02AE 1333 BSBW POLL_RSPRING ; This call has the effect of clearing
               02B1 1334 ; the UCB$M_PU_FRKBSY bit in
               02B1 1335 ; UCB$W_DEVSTS and also of finding any
               02B1 1336 ; responses that may have backed up
               02B1 1337 ; due to the bit's having been set.
50 01 D0 02B1 1338 MOVL #SS$NORMAL,R0 ; Set return code.
               02B4 1339 15$:
               05 02B4 1340 RSB
               02B5 1341
               02B5 1342 POWER_INIT:
               02B5 1343
01 01 E3 02B5 1344 BBCS #UCB$V_PU_FRKBSY,- ; Appropriate UCB fork block if free,
01 68 A5 02B7 1345 RSB UCB$W_DEVSTS(R5),20$ ; else continue and RSB.
               05 02BA 1346 ; If fork block busy, rely on
               02BB 1347 ; Interrupt Service Routine to GOTO
               02BB 1348 ; POST_POWER_FORK.
               02BB 1349 20$:
               02BB 1350 IOFORK ; Lower IPL so as to continue with
               02C1 1351 ; signaling SYSAPS that the VC fell.
18 11 02C1 1352 BRB 30$ ; Branch around Fork block appropriation.
               02C3 1353 POST_POWER_FORK:
               02C3 1354
               02C3 1355 ; Here we turn off CRB wakeup mechanism.
               02C3 1356
50 24 A5 D0 02C3 1357 MOVL UCB$L_CRB(R5),R0 ; R0 => CRB.
18 A0 01 CE 02C7 1358 MNEGL #1,CRB$L_DUE TIME(R0) ; Set infinite time.
FE2A CF 9E 02CB 1359 MOVAB NULL_ROUTINE,- ; Inocuous routine for now.
1C A0 02CF 1360 CRB$L_TOUTROUT(R0)
               02D1 1361
               05 01 E3 02D1 1362 BBCS #UCB$V_PU_FRKBSY,- ; Appropriate the UCB fork block if NOT
05 68 A5 02D3 1363 UCB$W_DEVSTS(R5),30$ ; busy and branch around.
10 A8 02D6 1364 B1SW #UCB$M_PU_MRESET,- ; Indicate that whoever has fork block
68 A5 02D8 1365 RSB UCB$W_DEVSTS(R5) ; busy, should reset upon wakeup.
               05 02DA 1366 ; Return to caller to kill thread.
               02DB 1367 30$:
10 AA 02DB 1368 B1CW #UCB$M_PU_MRESET,- ; Clear bit possibly set above.

```



```

+ CONTROLLER INIT

68 A5      02DD 1369
           AA 02DF 1370
           02E0 1371
64 A5 30    02E0 1372
           02E3 1373
           02E3 1374
           02E3 1375
           02E3 1376
           02E3 1377
           02E6 1378
53 03 D0    02E6 1379
           DD 02EB 1380
           DD 02ED 1381
53 00E4 C443 D0 02EF 1382
           OC B3 16 02F5 1383
           02F8 1384
           02F8 1385
           02F8 1386
           55 8ED0 02F8 1387
           53 8ED0 02FB 1388
E5 53 F4 02FE 1389
           99 11 0301 1390
           0303 1391

           BICW UCB$W_DEVSTS(R5)
           #UCB$M_ONLINE!-
           UCB$M-POWER,-
           UCB$W_STS(R5)
           ; If here we are not online,
           ; and the power failure is being handled

           ; Here we call the CONNECTION error routine for each connection.

           MOVL #3,R3
           40$:
           MOVL UCB$L_PDT(R5),R4
           PUSH R3
           PUSH R5
           MOVL PDT$L_PU_CDTARY(R4)[R3],R3
           JSB @CDT$L_ERRADDR(R3)
           ; R4 => PDT.
           ; Save connection index.
           ; Save R5 => UCB.
           ; R3 => CDT.
           ; Call connection error routine.
           ; NOTE: that for closed connections,
           ; the error routine is the NULL
           ; one that is just an RSB.
           ; Restore R5 => UCB.
           ; Restore R3 = connection index.
           ; Loop thru all possible indices.
           ; Branch back to re-init port.

           POPL R5
           POPL R3
           SOBGEQ R3,40$
           BRB 10$
           .dsabl lsb

```

+ Build PDT

00000000

0303 1393
0303 1394
0303 1395
0303 1396
0303 1397
0303 1398
0303 1399
0303 1400
0303 1401
0303 1402
0303 1403
0303 1404
0303 1405
0303 1406
0303 1407
0303 1408
0303 1409
0303 1410
0303 1411
0303 1412
0303 1413
0303 1414
0303 1415
0303 1416
0303 1417
0303 1418
0303 1419
0303 1420
0303 1421
0303 1422
0303 1423
0303 1424
0303 1425
0303 1426
0303 1427
0303 1428
0303 1429
0303 1430
0303 1431
0303 1432
0303 1433
0303 1434
0303 1435
0303 1436
0303 1437
0303 1438
0303 1439
0303 1440
0303 1441
0303 1442
0303 1443
0303 1444
0303 1445
0303 1446
0303 1447
0303 1448
0303 1449

.SBTTL + Build PDT

:+ BUILD_PDT - allocate and fill in non variable data in the UDA port PDT.
This routine has processor dependent code to separate code streams for
uVAX I ports, and for all other systems. The uVAX I needs special
treatment for various reasons.

1. First, for uVAX I, the communications area, the buffers, and the
Pseudo Map Registers (these are not needed at all for other systems)
must all be in physically contiguous memory pages. To accomodate
this, for uVAX I we allocate the entire PDT in physically
contiguous memory.
 2. Second, the base of the Pseudo Map Registers must be on a page
boundary. To accomodate this need, the first four pages of the
contiguous region we allocate for the PDT is used for this purpose.
A pointer to this region is left in PDT\$PQ_MAP.
 3. For uVAX I, the communications area must not straddle a 64KB
boundary. Currently, the communications area is much less
than a page in length and if it can be shown that the communications
area does not straddle a page boundary, this would be sufficient
to show that it does not straddle a 64KB boundary. Since the PDT
is located on a page boundary, we determine the byte offset of the
base of the communications area (symbol COMAREA_BOFF).
- COMAREA_BOFF = PDT\$COMAREA & 511
- This allows us to establish an ASSUME statement that proves
that the communications area does not straddle a page boundary.
- ASSUME COMAREA_BOFF+PDT\$COMAREALN LE 512
- Future changes to the PDT might break this ASSUME, and a possible
re-arrangement of the layout of the PDT would be called for at that
time.
4. Finally, the routines that implement SCS functions 'MAP IRP' and
'UNMAP' are different on uVAX I systems, so therefore the PDT
dispatch vectors for these functions are treated separately for
uVAX I.

Inputs:
R5
Outputs:
R0

-Addr of UCB
-Status code (SS\$NORMAL,SS\$INSFMEM)

PDT adjacency assumptions:

ASSUME PDT\$W_SIZE EQ 8
ASSUME PDT\$W_SIZE+2 EQ PDT\$B_TYPE

+ Build PDT

```

0303 1450 ASSUME PDT$B_TYPE+1 EQ PDT$B_SUBTYP
0303 1451 ASSUME PDT$B_SUBTYP+1 EQ PDT$C_SCSBASE
0303 1452
0303 1453 .enabl lsb
0303 1454
0303 1455 BUILD_PDT:
0303 1456
0303 1457 CPUDISP <<780,ALLOC_PDT_NOTUV1>,- ; Dispatch to allocate a CPU depende
0303 1458 <750,ALLOC_PDT_NOTUV1>,- ; sized PDT.
0303 1459 <730,ALLOC_PDT_NOTUV1>,-
0303 1460 <790,ALLOC_PDT_NOTUV1>,-
0303 1461 <UV1,ALLOC_PDT_UV1>,-
0303 1462 <8SS,ALLOC_PDT_8SS>,-
0303 1463 >
031D 1464
031D 1465
031D 1466 ALLOC_PDT_8SS: ; See if we are on a UNIBUS adapter
031D 1467 ; or else on a BDA.
50 24 A5 D0 031D 1468 MOVL UCBSL_CRB(R5),R0 ; R0 => CRB.
50 38 A0 D0 0321 1469 MOVL CRBSL_INTD+VECSL_ADP(R0),R0 ; R0 => ADP.
OE A0 01 B1 0325 1470 CMPW #ATS_OBA,ADPSW_ADPTYPE(R0) ; See if on UNIBUS adapter.
04 13 0329 1471 BEQL ALLOC_PDT_NOTUV1 ; If yes, branch.
032B 1472 ;
032B 1473 ; Here we must be on a BDA. Appropriate code may one day be placed here.
032B 1474 ; Till then,
032B 1475 BUG_CHECK UDAPORT,FATAL
032F 1476
032F 1477 ALLOC_PDT_NOTUV1:
032F 1478
51 0C94 8F 3C 032F 1479 MOVZWL #PDT$C_PULENGTH,R1 ; R1 contains PDT size needed for other
0334 1480 ; CPU's.
0955 30 0334 1481 BSBW ALLOC_POOL ; Allocate R1 bytes from pool
23 50 E8 0337 1482 BLBS R0,20$ ; Branch if success to common code.
0138 31 033A 1483 BRW 100$ ; Else goto return error
033D 1484
033D 1485 ALLOC_PDT_UV1: ; uVAX I needs physically contiguous
033D 1486 ; memory long enough for the map
033D 1487 ; registers (4 pages) and the PDT.
033D 1488
00001000 033D 1489 UV1_PDT_LENGTH = <PDT$C_UV1LENGTH + 511> & <^c511>
00000008 033D 1490 UV1_PDT_PAGES = UV1_PDT_LENGTH / 512 ; Pages needed for uVAX I PDT.
033D 1491
51 08 D0 033D 1492 MOVL #UV1_PDT_PAGES,R1 ; R1 contains pages needed for uVAX I
0340 1493 ; PDT.
0340 1494
00000000'GF 16 0340 1495 JSB G^EXESALOPHYCNTG ; Allocate physically contiguous memory.
03 50 E8 0346 1496 BLBS R0,10$ ; LBS means successful allocation.
0129 31 0349 1497 BRW 100$ ; Else goto error return.
034C 1498 10$:
00 6E 3F BB 034C 1499 PUSHR #^M<R0,R1,R2,R3,R4,R5> ; Save MOVC registers
1000 8F 2C 034E 1500 MOVCS #0,(SP),#0,- ; Zero initialize structure
62 0352 1501 #<UV1_PDT_PAGES>*512,-
3F BA 0355 1502 (R2)
0356 1503 POPR #^M<R0,R1,R2,R3,R4,R5> ; Restore MOVC registers
0358 1504
51 1000 8F B0 0358 1505 MOVW #UV1_PDT_LENGTH,R1 ; Set size of uVAX I PDT into R1.
035D 1506 20$:

```

+ Build PDT

0084	C5	52	D0	035D	1507	MOVL	R2,UCB\$PDT(R5)	:	Save PDT addr	
00DC	C2	55	D0	0362	1508	MOVL	R5,PDT\$PDT(UCB(R2))	:	PDT => UCB.	
53	24	A5	D0	0367	1509	MOVL	UCB\$PDT(CRB(R5),R3)	:	Get CRB addr	
10	A3	52	D0	036B	1510	MOVL	R2,CRB\$PDT_AUXSTRUC(R3)	:	and save PDT addr in CRB	
		82	7C	036F	1511	CLRQ	(R2)+	:	Init PDT, unused longwds,	
		01	B0	0371	1512	MOVW	#PDT\$M_SINGLHOST,-	:	Indicate port to single host bus, (-8	
	FC	A2		0373	1513		PDT\$PDT_PORTCHAR-8(R2)	:	takes into account (R2)+ above).	
		02	90	0375	1514	MOVB	#PDT\$PDT_PU,-	:	Indicate type of PDT created, (-8	
	FF	A2		0377	1515		PDT\$PDT_PDT_TYPE-8(R2)	:	takes into account (R2)+ above).	
				0379	1516					
				0379	1517	ASSUME	PDT\$PDT_SIZE	EQ	8	
	82	51	B0	0379	1518	MOVW	R1,(R2)+	:	PDT size,	
				037C	1519					
				037C	1520	ASSUME	PDT\$PDT_TYPE	EQ	10	
				037C	1521	ASSUME	PDT\$PDT_SUBTYPE	EQ	11	
82	0560	8F	B0	037C	1522	MOVW	#<DYN\$SCSCS_PDT@8 + DYN\$SCSCS>,(R2)+	:	structure subtype and type	
				0381	1523					
53	30	A3	D0	0381	1524	MOVL	CRB\$PDT_INTD+VEC\$PDT_INITIAL(R3),R3	:	Get addr of controller	
				0385	1525			:	init routine	
51	FE5B	CF	3E	0385	1526	MOVW	PUS\$SCSOFFSET,R1	:	Get addr of table of offsets	
				038A	1527			:	to SCS entries in PUDRIVER	
				038A	1528					
	50	81	32	038A	1529	30\$:	CVTWL	(R1)+,R0	:	Get offset to next SCS routine
		06	13	038D	1530		BEQL	40\$:	Branch if no more
82	53	50	C1	038F	1531		ADDL3	R0,R3,(R2)+	:	Add offset from controller init
				0393	1532			:	to addr of controller init	
				0393	1533			:	and store in PDT	
		F5	11	0393	1534	BRB	30\$:	Get next offset	
				0395	1535					
				0395	1536	40\$:				
51	0084	C5	D0	0395	1537	MOVL	UCB\$PDT(R5),R1	:	R1 => PDT.	
50	24	A5	D0	039A	1538	MOVL	UCB\$PDT(CRB(R5),R0)	:	R0 => CRB.	
	38	A0	D0	039E	1539	MOVL	CRB\$PDT_INTD+VEC\$PDT_ADP(R0),-	:	Save ADP address in PDT.	
	00E0	C1		03A1	1540		PDT\$PDT_ADP(R1)			
				03A4	1541	ASSUME	IDB\$PDT_CSR	EQ	0	
	2C	B0	D0	03A4	1542	MOVL	@CRB\$PDT_INTD+VEC\$PDT_IDB(R0),-	:	Save CSR addr in PDT.	
	0100	C1		03A7	1543		PDT\$PDT_PU_CSR(R1)			
00BC	C1	FE00	8F	3C	03AA	MOVZWL	#<127*512>,-	:	Define maximum byte count supported	
				03B1	1545		PDT\$PDT_MAXBCNT(R1)	:	for block transfers as 127 blocks.	
				03B1	1546					
				03B1	1547					
				03B1	1548	CPUDISP	<<780,NO_OVERLAYMAP>,-	:	Dispatch around overlaying of MAP	
				03B1	1549		<750,NO_OVERLAYMAP>,-	:	and UNMAP pointers uVAX I.	
				03B1	1550		<730,NO_OVERLAYMAP>,-			
				03B1	1551		<790,NO_OVERLAYMAP>,-			
				03B1	1552		<UV1,OVERLAYMAP>,-			
				03B1	1553		<8SS,OVERLAYMAP_8SS>,-			
				03B1	1554		>			
				03CB	1555					
				03CB	1556	OVERLAYMAP_8SS:		:	If on a UNIBUS adapter, goto	
				03CB	1557			:	NO_OVERLAYMAP.	
50	00E0	C1	D0	03CB	1558	MOVL	PDT\$PDT_ADP(R1),R0	:	R0 => ADP.	
	01	B1		03D0	1559	CMPW	#AT\$UBA,-	:	See if on UNIBUS adapter.	
	0E	A0		03D2	1560		ADP\$PDT_ADP_TYPE(R0)			
	1A	13		03D4	1561	BEQL	NO_OVERLAYMAP	:	EQL implies UNIBUS adapter.	
				03D6	1562					
				03D6	1563			:	For the BDA port the MAPIRP and UNMAPIRP functions are distinct.	

+ Build PDT

```
03D6 1564 ; Here we overlay the PDT dispatch vectors for these functions.
03D6 1565
112F'CF 9E 03D6 1566 MOVAB W^FPC$MAPIRP_BDA,- ; Overlay PDT dispatch vector for MAPIRP
34 A1 03DA 1567 PDT$M_MAPIRP(R1) ; on BDA.
11BD'CF 9E 03DC 1568 MOVAB W^FPC$UNMAP_BDA,- ; Overlay PDT dispatch vector for UNMAP
64 A1 03E0 1569 PDT$M_UNMAP(R1) ; on BDA.
OC 11 03E2 1570 BRB NO_OVERLAYMAP ; Branch around to join common code.
03E4 1571
03E4 1572 OVERLAYMAP:
03E4 1573
03E4 1574 ; For the uVAX I Q-BUS port the MAPIRP and UNMAPIRP functions are distinct.
03E4 1575 ; Here we overlay the PDT dispatch vectors for these functions.
03E4 1576
1014'CF 9E 03E4 1577 MOVAB W^FPC$MAPIRP_UV1,- ; Overlay PDT dispatch vector for MAPIRP
34 A1 03E8 1578 PDT$M_MAPIRP(R1) ; on uVAX I.
1162'CF 9E 03EA 1579 MOVAB W^FPC$UNMAP_UV1,- ; Overlay PDT dispatch vector for UNMAP
64 A1 03EE 1580 PDT$M_UNMAP(R1) ; on uVAX I.
03F0 1581
03F0 1582 NO_OVERLAYMAP:
03F0 1583
03F0 1584 ; Init NULL CDT with addresses of message and datagram input routines.
03F0 1585
FCE8 CF 9E 03F0 1586 MOVAB NULL_MSG_INPUT,- ; Address of message input
FD05 CF 03F4 1587 NULL_CDT+CDT$M_MSGINPUT ; routine.
FCE5 CF 9E 03F7 1588 MOVAB NULL_DG_INPUT,- ; Address of datagram input
FD02 CF 03FB 1589 NULL_CDT+CDT$M_DGINPUT ; routine.
FCF7 CF 9E 03FE 1590 MOVAB NULL_ERR_ROUT,- ; Address of error routine.
FD03 CF 0402 1591 NULL_CDT+CDT$M_ERRADDR
FD2B CF 9E 0405 1592 MOVAB NULL_CDT+CDT$M_CRWAITQFL,- ; Init dummy list header.
FD28 CF 0409 1593 NULL_CDT+CDT$M_CRWAITQFL
FD24 CF 9E 040C 1594 MOVAB NULL_CDT+CDT$M_CRWAITQFL,-
FD25 CF 0410 1595 NULL_CDT+CDT$M_CRWAITQBL
0413 1596
0413 1597 ; Init CDT pointers to all point to NULL CDT.
0413 1598
50 FCE5 CF 9E 0413 1599 MOVAB NULL_CDT,R0 ; R0 => NULL CDT.
00E4 C1 50 D0 0418 1600 MOVL R0,PDT$M_PU_VCO(R1) ; VCO CDT pointer => NULL CDT.
00E8 C1 50 D0 041D 1601 MOVL R0,PDT$M_PU_VC1(R1) ; VC1 CDT pointer => NULL CDT.
00EC C1 50 D0 0422 1602 MOVL R0,PDT$M_PU_VC2(R1) ; VC2 CDT pointer => NULL CDT.
00F0 C1 50 D0 0427 1603 MOVL R0,PDT$M_PU_VC255(R1) ; VC255 CDT pointer => NULL CDT.
042C 1604
01 FCA8 CF D1 042C 1605 CMPL PUSL_TRACE_VARIABLE,#1 ; Determine what kind (if any) tracing
0431 1606 ; is desired. 0 implies no tracing,
0431 1607 ; 1 implies only try to allocate
0431 1608 ; resources for "PT" tracing, and
0431 1609 ; 2 implies always try to allocate.
0431 1610
3F 19 0431 1611 BLSS 90$ ; LSS implies 0, so branch around since
0433 1612 ; we want no tracing.
0B 14 0433 1613 BGTR 80$ ; GTR implies 2, so branch around to try
0435 1614 ; to allocate trace tables.
0435 1615
0435 1616 ; If here we only want "PT" tracing,
0435 1617 ; then test for a "PT" port and
0435 1618 ; if not there, branch around.
0435 1619
51 28 A5 D0 0435 1620 MOVL UCBS$L_DDB(R5),R1 ; R1 => DDB. See if "PU" or "PT" device
```


+ TRACE_COMMAND and TRACE_RESPONSE

.SBTTL + TRACE_COMMAND and TRACE_RESPONSE

0476 1645
0476 1646
0476 1647 ; Routines to record command and response buffer contents in the trace table.
0476 1648 ; Trace table entries are 96 bytes long so that they line up nicely in
0476 1649 ; a dump.
0476 1650 ;
0476 1651

TRACE_COMMAND:

; INPUTS: R2 => Command buffer,
; R4 => PDT.

7E 50 7D 0476 1655 MOVQ R0,-(SP)
50 52 D0 0479 1656 MOVL R2,R0
06 11 047C 1657 BRB TRACE_COMMON

; Save R0 and R1.
; R0 => buffer to trace.
; Branch around to common code.

TRACE_RESPONSE:

; INPUTS: R3 => Response buffer,
; R4 => PDT.

7E 50 7D 047E 1662 MOVQ R0,-(SP)
50 53 D0 0481 1663 MOVL R3,R0

; Save R0 and R1.
; R0 => buffer to trace.

TRACE_COMMON:

0C88 C4 D5 0484 1667 TSTL PDT\$_TRTABLE(R4)
57 13 0488 1668 BEQL 30\$

; Test for existence of trace table.
; EQL implies unable to allocate table.

51 0C8C C4 D0 0490 1672 MOVL PDT\$_TRTBLPTR(R4),R1
51 0C90 C4 D1 0495 1673 CMPL PDT\$_TRTBLEND(R4),R1
05 14 049A 1674
51 0C88 C4 D0 049C 1676

; Prevent interrupts during allocation
; of trace table entry.
; R1 => area in trace table to use.
; See if we should circle back to start
; of trace table.
; GTR implies NO.
; R1 => base of trace table.

0C8C C4 51 00000060 8F C1 04A1 1677 20\$: ADDL3 #96,R1,PDT\$_TRTBLPTR(R4); Point to next entry.

; Undo DSBINT.
; Twelve long words are 96 bytes.

81 80 7D 04AB 1679 ENBINT
81 80 7D 04AE 1680 MOVQ (R0)+,(R1)+
81 80 7D 04B1 1681 MOVQ (R0)+,(R1)+
81 80 7D 04B4 1682 MOVQ (R0)+,(R1)+
81 80 7D 04B7 1683 MOVQ (R0)+,(R1)+
81 80 7D 04BA 1684 MOVQ (R0)+,(R1)+
81 80 7D 04BD 1685 MOVQ (R0)+,(R1)+
81 80 7D 04C0 1686 MOVQ (R0)+,(R1)+
81 80 7D 04C3 1687 MOVQ (R0)+,(R1)+
81 80 7D 04C6 1688 MOVQ (R0)+,(R1)+
81 80 7D 04C9 1689 MOVQ (R0)+,(R1)+
81 6E 7D 04CC 1690 MOVQ (SP),(R1)+
81 08 AE D0 04CF 1691 MOVQ 8(SP),(R1)+
50 00DC C4 D0 04D3 1692 MOVL PDT\$_UCB0(R4),R0
FC A1 0088 C0 C2 04D8 1693 MOVL UCB\$_DDT(R0),-4(R1)
81 01 CE 04DE 1694 MNEGL #1,(R1)+
50 8E 7D 04E1 1695 30\$: MOVQ (SP)+,R0
05 04E4 1697 RSB

; Trace saved R0, R1.
; Also trace caller's return point.
; R0 => UCB.
; Make traced return point relative.
; Flag marks end of trace entry.

; Restore R0 and R1.

+ INIT_UDA_BUFFERS

.SBTTL + INIT_UDA_BUFFERS

INIT_UDA_BUFFERS - accomplishes the following:

1. It fills in UCBSW_PU_BCNT, UCBSW_PU_BOFF and UCBSL_PU_SVAPTE and clears CRBSL_INTD+VECSW_MAPREG. It then copies them to DCBSW_BCNT, UCBSW_BOFF and UCBSL_SVAPTE so that the standard UNIBUS map register allocation routine will allocate enough of these map registers to map the area located at PDTSL_COMAREA which is PDTSC_RINGLEN long. It then calls IOCSALOUBMAPRM to permanently allocate the map registers. The value left in UCBSL_PU_SVAPTE, along with the values left in CRBSL_INTD+VECSW_MAPREG by IOCSALOUBMAPRM, allow the later loading of the allocated map registers. (Note the map registers are loaded at initialization time and after every reset of the port.)
2. It calculates the UNIBUS virtual address of the text portion of each of the 2*UDASK_RINGSIZE buffers and stores this virtual address in the low order 18 bits of the buffer header, at offset UDABSL_DESCRIP. The UDA_M_OWN and UDA_M_FLAG bits are also set on in UDABSL_DESCRIP so that this value can now be moved directly into a ring slot (be it command or response ring) to effect the transfer of this buffer to the port.

Inputs:

R5

-Addr of UCB

Outputs:

R0

R1-R4

R5

-Status code (SSS_NORMAL or 0 if no SVPN)

-Destroyed

-Preserved

.enabl lsb

INIT_UDA_BUFFERS:

54 0084 C5 D0

MOVCL UCBSL_PDT(R5),R4

; R4 => PDT.

04EA 1740

CPUDISP <<780,COM_INIT_UDA_BUFS>,-

; Dispatch to allow special
; casing of uVAX I.

04EA 1741

<750,COM_INIT_UDA_BUFS>,-

04EA 1742

<730,COM_INIT_UDA_BUFS>,-

04EA 1743

<790,COM_INIT_UDA_BUFS>,-

04EA 1744

<UV1,INIT_UDA_BUFS_UV1>,-

04EA 1745

<8SS,INIT_UDA_BUFS_8SS>,-

04EA 1746

>

0504 1747

0504 1748

INIT_UDA_BUFS_8SS:

50 00E0 C4 D0

MOVCL PDTSL_ADP(R4),R0

; R0 => ADP.

0509 1751

CMPW #ATS_OBA,-

; See if on UNIBUS adapter.

OE A0

050B 1752

ADPSW ADPTYPE(R0)

03

050D 1753

BEQL COM_INIT_UDA_BUFS

; EQL implies UNIBUS adapter.

01BD

050F 1754

BRW INIT_BDA_BUFS

; Else branch to init BDA buffers.

0512 1755

+ INIT_UDA_BUFFERS

```
0512 1756 COM_INIT_UDA_BUFS:
0512 1757
0512 1758 ; Initialize fields in UCB so as to allow standard IOSUBNPAG routines
0512 1759 ; to allocate MAP registers.
0512 1760
0512 1761 MOVW #PDT$C_RINGLEN,- ; Length of area to map.
0516 1762 UCBSW_PU_BCNT(R5)
0519 1763 MOVAB PDT$C_COMAREA(R4),R2 ; R2 => area to map.
051E 1764 BICW3 #^XFE00,R2,-
0526 1765 UCBSW_PU_BOFF(R5) ; Get offset of area to map.
0526 1766 EXTZV S^#VASS_VPN,-
0528 1767 S^#VASS_VPN,R2,R2 ; R2 = virtual page # of area.
052B 1768 MOVL G^MMG$GC_SPTBASE,R0 ; R0 => system page table.
0532 1769 MOVAL (R0)[R2],UCBSL_PU_SVAPTE(R5)
0538 1770
0538 1771
0538 1772 MOVL UCBSL_CRB(R5),R3 ; R3 => CRB.
053C 1773 CLRL CRBSL_INTD+VECSW_MAPREG(R3)
053F 1774 CPUDISP <<780,DATAPATH_780>,- ; Initialize to direct datapath (0).
053F 1775 <<750,DATAPATH_750>,- ; Label of VAX-11/780 specific code.
053F 1776 <<730,DATAPATH_730>,- ; Label of VAX-11/750 specific code.
053F 1777 <<790,DATAPATH_790>,- ; Label of VAX-11/730 specific code.
053F 1778 <<8SS,DATAPATH_8SS>,- ; Label of VAX-11/790 specific code.
053F 1779 >
053F 1780
0555 1781 DATAPATH_8SS:
0555 1782 MOVAB REQDATAP_8SS,- ; Overlay transfer vector when running
0559 1783 REQDATAPATH_IV ; on a VAX-11/8SS.
055C 1784 MOVAB RELDATAP_8SS,- ; Overlay transfer vector when running
0560 1785 RELDATAPATH_IV ; on a VAX-11/8SS.
0563 1786 BRB DATAPATH_780 ; And branch around.
0565 1787
0565 1788 DATAPATH_750:
0565 1789 MOVAB REQDATAP_750,- ; Overlay transfer vector when running
0569 1790 REQDATAPATH_IV ; on a VAX-11/750.
056C 1791 MOVAB RELDATAP_750,- ; Overlay transfer vector when running
0570 1792 RELDATAPATH_IV ; on a VAX-11/750.
0573 1793 BRB DATAPATH_780 ; And branch around.
0575 1794
0575 1795 DATAPATH_730:
0575 1796 MOVAB REQDATAP_730,- ; Overlay transfer vector when running
0579 1797 REQDATAPATH_IV ; on a VAX-11/730.
057C 1798 MOVAB RELDATAP_730,- ; Overlay transfer vector when running
0580 1799 RELDATAPATH_IV ; on a VAX-11/730. Then fall thru.
0583 1800
0583 1801 DATAPATH_780:
0583 1802 DATAPATH_790: ; 11/790 mimics the 11/780.
0583 1803 ASSUME PDT$B_BDPUSECNT EQ PDT$B_DATAPATH+1
0583 1804 CLRW PDT$B_DATAPATH(R4) ; Clear # of semi-perm BDP and use count.
0587 1805
0587 1806 CLRB CRBSL_INTD+VECSB_DATAPATH(R3) ; Clear CRB field.
058A 1807
058A 1808 ASSUME UCBSW_BOFF EQ UCBSL_SVAPTE+4
058A 1809 ASSUME UCBSW_BCNT EQ UCBSW_BOFF+2
058A 1810 ASSUME UCBSW_PU_BOFF EQ UCBSL_PU_SVAPTE+4
058A 1811 ASSUME UCBSW_PU_BCNT EQ UCBSW_PU_BOFF+2
058A 1812
```

+ INIT_UDA_BUFFERS

```
00BC C5 7D 058A 1813      MOVQ   UCB$$_PU_SVAPTE(R5),-      ; Copy parameters to standard
78 A5      058E 1814      UCB$$_SVAPTE(R5)      ; locations.
00000000 GF 16 0590 1815      JSB     G*IOC$ALOUBMAPRM      ; Permanently allocate map registers.
                                0596 1816
                                0596 1817 ; Here calculate the UNIBUS virtual address of each buffer.
                                0596 1818
50 00C0 C5 3C 0596 1819      MOVZWL  UCB$$_PU_BOFF(R5),R0      ; Byte offset of mapped area.
51 24 A5 D0 059B 1820      MOVL     UCB$$_CRB(R5),R1      ; R1 => CRB
                                059F 1821
50 09 09 34 A1 F0 059F 1822      INSV    CRB$$_INTD+VEC$$_MAPREG(R1),#9,#9,R0      ; R0 contains UNIBUS
                                05A5 1823      ; VA of base of area.
                                05A5 1824
                                50 DD 05A5 1825      PUSHL   R0      ; Save on stack.
                                50 D4 05A7 1826      CLRL    R0      ; Initialize loop counter.
01 029C C4 9E 05A9 1828      MOVAB   PDT$$_PU_BUFARY+UDAB$$_TEXT(R4),R1      ; R1 => text portion
                                05AE 1830      ; of first buffer.
                                05AE 1831 5$:
                                05AE 1832 ;
F6 A1 50 9B 05AE 1833      ASSUME   UDAB$$_BUFFNO+1      is Reserved byte.
                                05B2 1834      MOVZBW  R0,UDAB$$_BUFFNO-UDAB$$_TEXT(R1) ; Store index of this
                                05B2 1835      MOVAB   -UDAB$$_TEXT(R1),-      ; Point Buffer table array
014C C440 9E 05B5 1836      PDT$$_B$TABLE(R4)[R0]      ; element to this buffer.
52 0200 C4 9E 05B9 1837      MOVAB   PDT$$_COMAREA(R4),R2      ; R2 => base of mapped area.
52 51 52 C3 05BE 1838      SUBL3   R2,R1,R2      ; R2 = offset of this buffer
                                05C2 1839      ; from base of mapped area.
F8 A1 52 6E C1 05C2 1840      ADDL3   (SP),R2,-      ; Calculate UNIBUS VA of this
                                05C7 1841      UDAB$$_DESCRIP-UDAB$$_TEXT(R1) ; buffer.
                                05C7 1842
                                C8 05C7 1843      BISL    #UDA_M_OWN!-      ; Set port ownership and mark
                                05C8 1844      UDA_M_FLAG,-      ; so port should interrupt on
F8 A1 C0000000 8F 05C8 1845      UDAB$$_DESCRIP-UDAB$$_TEXT(R1) ; ring transitions.
                                05CF 1846
51 00000050 8F C0 05CF 1847      ADDL    #UDAB$$_LENGTH,R1      ; R1 => text portion of next buffer.
D4 50 20 F2 05D6 1848      AOBLSS  #2*UDAB$$_RINGSIZE,R0,5$ ; Loop thru all buffers.
                                05DA 1849
                                50 8E D5 05DA 1850      TSTL    (SP)+      ; Clear stack.
                                50 01 D0 05DC 1851      MOVL    #$$$NORMAL,R0      ; Set status return.
                                00F1 31 05DF 1852      BRW     INIT_UDA_BUFS_RTN      ; Branch around to return.
                                05E2 1853
                                05E2 1854 INIT_UDA_BUFS_UV1:
                                05E2 1855
53 24 A5 D0 05E2 1856      MOVL    UCB$$_CRB(R5),R3      ; R3 => CRB.
34 A3 D4 05E6 1857      CLRL    CRB$$_INTD+VEC$$_MAPREG(R3) ; Initialize to direct datapath (0).
                                05E9 1858
                                05E9 1859
                                05E9 1860 ; To use the common allocate/deallocate map register routines, the ADP$$_CSR
                                05E9 1861 ; field in the Qbus adapter control block must be set to point to a base address
                                05E9 1862 ; such that the pseudo map registers existing in the PDT appear at the UBA$$_MAP
                                05E9 1863 ; offset from this base address. Currently the SVAPTEs allocated at the time
                                05E9 1864 ; the adapter is initialized (INIADP) point to non-existent memory addresses.
                                05E9 1865 ; These must instead point to the 4 pages of map entries allocated as part
                                05E9 1866 ; of the PDT.
                                05E9 1867
                                00 E2 05E9 1868      BBSS    #MAP$$_MAPREGS,-      ; Ensure that we use only one set of
FAE2 CF 05EB 1869      PUSL_DRIVER_STS,-      ; "map registers", gets mapped into
```


+ INIT_UDA_BUFFERS

F 11

16-SEP-1984 01:05:05 VAX/VMS Macro V04-00
5-SEP-1984 00:17:10 [DRIVER.SRC]PUDRIVER.MAR;1

Page 42
(2)

53			05EE	1870	15\$; the virtual range pointed at by
			05EF	1871			; ADP\$CSR. This bit allows us to
			05EF	1872			; pass thru here just once, for the
			05EF	1873			; first UQPORT found on a UV1 system.
			05EF	1874			
51	04	D0	05EF	1875	MOVL	#4,R1	; R1 contains pages needed for uVAX I
			05F2	1876			; pseudo map registers.
			05F2	1877			
00000000	'GF	16	05F2	1878	JSB	G^EXESALOPHYCNTG	; Allocate physically contiguous memory.
08	50	E8	05F8	1879	BLBS	R0,8\$; LBS means successful allocation.
	01	CA	05FB	1880	BICL	#MAP\$M MAPREGS,-	; Undo setting of bit above since we
FAD0	CF		05FD	1881		PUSL DRIVER ST\$; have failed to create map registers.
	00D0	31	0600	1882	BRW	INIT_UDA_BUFS_RTN	; And goto error return.
			0603	1883			
0C94	C4	52	0603	1884	MOVL	R2,PDT\$PQ_MAP(R4)	; Remember address of map registers.
	51	52	0608	1885	MOVL	R2,R1	; Setup for call to BSBW to convert
			060B	1886			; virtual address of map registers
			060B	1887			; to physical address.
00C6		30	060B	1888	BSBW	VIRT_TO_PHYAD	; Convert R1 to physical address and
			060E	1889			; leave result on top of stack.
			060E	1890			
50	50	38	060E	1891	ASSUME	ADP\$CSR EQ 0	
			060E	1891	MOVL	@CRB\$C_INTD+VECSL ADP(R3),R0	; Get address of Qbus adapter CSR.
51	50	15	0612	1892	EXTZV	#VASV VPN,#VASS VPN,R0,R0	; Convert from VA to VPN.
	00000000	'GF	0617	1893	MOVL	G^MMG\$GL_SPTBASE,R1	; Get address of system page table.
	50	6140	061E	1894	MOVAL	(R1)[R0],R0	; Get SVAPTE of Qbus adapter CSR.
			0622	1895	ADDL	#<4*4>,R0	; Point to SVAPTEs for map registers.
51	6E	F7	0625	1896	ASHL	#-VASV VPN,(SP),R1	; Get PFN of pseudo map registers.
			062A	1897	MOVZBL	#4,-(SP)	; Count of pages of map registers.
60	15	00	062D	1898	INSV	R1,#PTESV_PFN,#PTES\$PFN,(R0)	; Set SPTTE to map pseudo map reg.
			0632	1899	ADDL	#4,R0	; Get SVAPTE of next PTE.
			0635	1900	INCL	R1	; Point to next physical PFN.
	F3	6E	0637	1901	SOBCTR	(SP),10\$; Loop to map all four pages.
	5E	04	063A	1902	ADDL	#4,SP	; Clean counter off stack.
			063D	1903			
FA92	CF	8ED0	063D	1904	POPL	PUSL_MPHYAD	; Pop physical address into static.
			0642	1905			
			0642	1906			
0129	C4	B4	0642	1907	ASSUME	PDT\$B_BDPUSECNT EQ	PDT\$B_DATAPATH+1
			0646	1908	CLRW	PDT\$B_DATAPATH(R4)	; Clear # of semi-perm BDP and use count.
	50	D4	0646	1909	CLRL	R0	; Initialize loop counter.
			0648	1910			
51	029C	C4	0648	1911	MOVAB	PDT\$P_PU_BUFARY+UDAB\$T_TEXT(R4),R1	; R1 => text portion
			064D	1912			; of first buffer.
			064D	1913			
			064D	1914			
F6	A1	50	064D	1915	ASSUME	UDAB\$B_BUFFNO+1 is	Reserved byte.
			0651	1916	MOVZBW	R0,UDAB\$B_BUFFNO-UDAB\$T_TEXT(R1)	; Store index of this
			0651	1917			; buffer in its header.
EC	A1	9E	0651	1917	MOVAB	-UDAB\$T_TEXT(R1),-	; Point Buffer table array
014C	C440		0654	1918		PDT\$P_B\$TABLE(R4)[R0]	; element to this buffer.
			0658	1919			
0079		30	0658	1920	BSBW	VIRT_TO_PHYAD	; Convert R1 to physical address and
			065B	1921			; leave result on top of stack.
8E	C00C0000	8F	065B	1922	BISL3	#UDA_M_OWN!-	; Set port ownership and mark
			0662	1923		UDA_M_FLAG,(SP)+,-	; so port should interrupt on
	F8	A1	0662	1924		UDAB\$P_DESCRIP-UDAB\$T_TEXT(R1)	; ring transitions.
			0664	1925			
51	00000050	8F	0664	1926	ADDL	#UDAB\$C_LENGTH,R1	; R1 => text portion of next buffer.

+ INIT_UDA_BUFFERS

```
DE 50 20 F2 066B 1927 AOBLS #2*UDASK_RINGSIZE,R0,20$; Loop thru all buffers.
066F 1928
066F 1929 : Here initialize the Aligned Page and its accompanying items. These are:
066F 1930 :
066F 1931 : 1. PDT$PQ_PGQFL and PDT$PQ_PGQBL - An allocation queue header
066F 1932 : wherein we queue CDRP's waiting waiting for this resource.
066F 1933 :
066F 1934 : 2. PDT$PQ_POWNER - The CDRP of the current owner of the Aligned Page.
066F 1935 :
066F 1936 : 3. PDT$PQ_PGPHAD - The Physical address of PDT$PQ_ALGNPG.
066F 1937 :
066F 1938 :
0C98 C4 9E 066F 1939 MOVAB PDT$PQ_PGQFL(R4),- ; Set up Queue Header.
0C98 C4 0673 1940 PDT$PQ_PGQFL(R4)
0C98 C4 9E 0676 1941 MOVAB PDT$PQ_PGQFL(R4),-
0C9C C4 067A 1942 PDT$PQ_PGQBL(R4)
067D 1943
0CA0 C4 D4 067D 1944 CLRL PDT$PQ_POWNER(R4) ; Initialize to NO owner.
51 0CB4 C4 9E 0681 1945 MOVAB PDT$PQ_ALGNPG(R4),R1 ; R1 => Aligned Page.
004B 30 0686 1946 BSBW VIRT_TO_PHYAD ; Convert R1 to physical address and
0689 1947 ; leave result on top of stack.
0CA4 C4 8ED0 0689 1948 POPL PDT$PQ_PGPHAD(R4) ; Save Aligned Page physical address.
068E 1949
068E 1950 : Since the INIT logic only allocates a system virtual page for the class
068E 1951 : driver, we must allocate a page for the boot driver in its initialization
068E 1952 : routine. We initialize the PDT fields associated with the System Virtual Page
068E 1953 : that we use to map the user's non-word aligned buffer so that we can copy
068E 1954 : it to the Aligned Page. The fields are PDT$PQ_UBFSVA that we initialize
068E 1955 : to be the System Virtual Address of this page; and PDT$PQ_SVPSTE that
068E 1956 : we set to be the virtual address of the PTE that describes this page.
068E 1957
068E 1958 MOVZBL #1,R1 ; Request one SVPN.
00000000'GF 16 0691 1959 JSB G^IOCSALLOSPT ; Allocate the system virtual page.
01 50 E8 0697 1960 BLBS R0,30$ ; Br if successful allocation.
069A 1961 RSB ; Return error to caller.
51 00000000'9F D0 069B 1962 30$: MOVL @MMG$GL_SPTBASE,R1 ; Get address of SPT (PIC)
901FFFFFF 8F D0 06A2 1963 MOVL #<PTE$C_RW!PTE$M_VALID!PTE$M_PFN>,- ; Set SPT to valid,
6142 06A8 1964 (R1)[R2] ; writable, and non-existent PFN.
74 A5 52 D0 06AA 1965 MOVL R2,UCB$S_SVPN(R5) ; Get Virtual Page Number of page.
51 52 09 78 06AE 1966 ASHL #9,R2,R1 ; Multiply by 512 to get relative address
OCA8 C4 51 80000000 8F C9 06B2 1967 BISL3 #1@31,R1,- ; in system space. Then set high
06BC 1968 PDT$PQ_UBFSVA(R4) ; order bit to make system address.
53 00000000'GF D0 06BC 1969 MOVL G^MMG$GL_SPTBASE,R3 ; R3 => system page table.
OCAC C4 6342 DE 06C3 1970 MOVAL (R3)[R2],- ; Calcualte address of PTE slot that
06C9 1971 PDT$PQ_SVPSTE(R4) ; describes this page.
50 01 D0 06C9 1972 MOVL #SS$_NORMAL,R0 ; Set return code.
0004 31 06CC 1973 BRW INIT_UDA_BUFS_RTN ; Branch around to common return.
06CF 1974
06CF 1975 INIT_BDA_BUFS:
06CF 1976 BUG_CHECK UDAPORT,FATAL ; For now only a place holder.
06D3 1977
06D3 1978 INIT_UDA_BUFS_RTN: ; Common return label.
06D3 1979
05 06D3 1980 RSB ; Return to caller
06D4 1981 .dsabl lsb
```


+ VIRT_TO_PHYAD

.SBTTL + VIRT_TO_PHYAD

06D4 1983
06D4 1984
06D4 1985
06D4 1986
06D4 1987
06D4 1988
06D4 1989
06D4 1990
06D4 1991
06D4 1992
06D4 1993
06D4 1994
06D4 1995
06D4 1996
06D4 1997
06D4 1998
06D4 1999
06D6 2000
06D6 2001
06D8 2002
06E0 2003
06E2 2004
06E5 2005
06EC 2006
06F0 2007
06F0 2008
06F2 2009
06F5 2010
06F9 2011
06FE 2012
0700 2013

VIRT_TO_PHYAD - routine to convert a System Virtual Address into a physical address.

Inputs:
R1 = System Virtual Address to convert

Outputs:
Physical Address left on top of stack.
All registers preserved.

VIRT_TO_PHYAD:

50	51	FFFFFE00	0F	BB	06D4	1999	PUSHL	(SP)	:	Create slot at top of stack for return
			8F	CB	06D6	2000			:	value.
	52	51	15	EF	06D6	2001	PUSHR	#^M<R0,R1,R2,R3>	:	Save registers.
			09		06D8	2002	BICL3	#-512,R1,R0	:	R0 = Boif of address.
	53	00000000	GF	D0	06E0	2003	EXTZV	#VASV_VPN,-	:	Extract VPN
		53	6342	D0	06E2	2004		#VASS_VPN,R1,R2	:	and put it in R2.
					06E5	2005	MOVL	G^MMG\$GL_SPTBASE,R3	:	R3 => system page table.
					06EC	2006	MOVL	(R3)[R2],R3	:	R3 => PTE.
			00	EF	06F0	2007			:	
	53	53	15		06F0	2008	EXTZV	#PTESV_PFN,-	:	Get page frame number of buffer
					06F2	2009		#PTESV_PFN,R3,R3	:	page into R3.
	53	53	09	78	06F5	2010	ASHL	#VASV_VPN,R3,R3	:	Shift into place for physical address.
14	AE	53	50	C9	06F9	2011	BISL3	R0,R3,20(SP)	:	Put result into stack slot.
			0F	BA	06FE	2012	POPR	#^M<R0,R1,R2,R3>	:	Restore registers.
				05	0700	2013	RSB		:	Return to caller.

+ INIT_INIT_STEPS

```
.SBTTL + INIT_INIT_STEPS
0701 2015
0701 2016
0701 2017 :+
0701 2018 : INIT_INIT_STEPS - initialize the host responses for UDA hardware
0701 2019 : initialization.
0701 2020 :
0701 2021 : Inputs:
0701 2022 : R5 -Addr of UCB
0701 2023 :
0701 2024 : Outputs:
0701 2025 : UCB$W_HOSTSTEPx fields all initialized
0701 2026 :
0701 2027 : Registers R0-R2 destroyed
0701 2028 :-
0701 2029 :
0701 2030 : .enabl lsb
0701 2031 :
0701 2032 INIT_INIT_STEPS:
0701 2033 :
0701 2034 : First calculate the vector for this UDA. This is done by scanning the
0701 2035 : UNIBUS adapter interrupt vector looking for a slot that points
0701 2036 : into this CRB$$_INTD.
0701 2037 :
50 24 A5 D0 0701 2038 MOVL UCB$$_CRB(R5),R0 ; R0 => CRB.
24 A0 DF 0705 2039 PUSHAL CRB$$_INTD(R0) ; Save interrupt dispatcher
0708 2040 ; address on stack.
50 2C A0 D0 0708 2041 MOVL CRB$$_INTD+VEC$$_IDB(R0),R0 ; R0 => IDB.
50 14 A0 D0 070C 2042 MOVL IDB$$_ADP(R0),R0 ; R0 => ADP.
50 10 A0 D0 0710 2043 MOVL ADP$$_VECTOR(R0),R0 ; R0 => UBA interrupt vectors.
0714 2044 :
51 D4 0714 2045 CLRL R1 ; Initialize counter.
0716 2046 10$:
52 80 D0 0716 2047 MOVL (R0)+,R2 ; R2 = contents of a vector.
52 03 CA 0719 2048 BICL #3,R2 ; Clear low order 2 bits to remove
071C 2049 ; processor (780 vs 750) differences.
6E 52 D1 071C 2050 CMPL R2,(SP) ; Is this our vector?
0D 13 071F 2051 BEQL 20$ ; EQL means yes.
ED 51 00000080 8F F2 0721 2052 AOBLSS #NUMUBAVEC,R1,10$ ; Else loop back to try again.
8E D5 0729 2053 TSTL (SP)+ ; It had better be there
50 D4 072B 2054 CLRL R0 ; or else we return with an error
05 072D 2055 RSB ; indication.
072E 2056 20$:
8E D5 072E 2057 TSTL (SP)+ ; Remove unneeded stack value.
0730 2058 :
51 A9 0730 2059 BISW3 R1,- ; Or in vector address/4
0732 2060 #HS1 M_IE!- ; (i.e. vector #) with
0732 2061 <UDASK_RINGEXP@HS1_V_RRNGLEN>!- ; Interrupt Enable bit, and
0732 2062 <UDASK_RINGEXP@HS1_V_CRNGLEN>!- ; the response and command ring
0732 2063 HS1 M_BIT15,- ; lengths and the high bit on,
0732 2064 UCB$$_HOSTSTEP1(R5) ; and save it in the UCB.
0738 2065 :
0738 2066 : Here we calculate the UNIBUS virtual address of the ring base so as to be
0738 2067 : able to communicate it to the controller.
0738 2068 :
0738 2069 :
0738 2070 CPUDISP <<780,RINGBASE_COM>,-
0738 2071 <<750,RINGBASE_COM>,-
```


+ INIT_INIT_STEPS

```
0738 2072 <730,RINGBASE_COM>,-
0738 2073 <790,RINGBASE_COM>,-
0738 2074 <UV1,RINGBASE_UV1>,-
0738 2075 <8SS,RINGBASE_8SS>,-
0738 2076 >
0752 2077
0752 2078 RINGBASE_8SS:
50 00E0 C4 D0 0752 2079 MOVL PDT$ADP(R4),R0 ; R0 => ADP.
01 B1 0757 2080 CMPW #AT$OBA,- ; See if on UNIBUS adapter.
OE A0 0759 2081 ADP$Q ADPTYPE(R0)
04 13 075B 2082 BEQL RINGBASE_COM ; EQL implies UNIBUS adapter.
075D 2083 BUG_CHECK UDAPORT,FATAL ; For now, if on BDA.
0761 2084
0761 2085 RINGBASE_COM:
0761 2086
50 00C0 C5 3C 0761 2087 MOVZWL UCBSW_PU BOFF(R5),R0 ; R0 contains byte offset of ringbase.
51 24 A5 D0 0766 2088 MOVL UCBSL_CRB(R5),R1 ; R1 => CRB.
076A 2089
50 09 09 34 A1 F0 076A 2090 INSV CRBSL_INTD+VECSW_MAPREG(R1),#9,#9,R0 ; R0 contains the UBA
0770 2091 ; virtual address of
0770 2092 ; the communications
0770 2093 ; area.
50 08 C0 0770 2094 ADDL #PDT$RINGBASE-PDT$COMAREA,R0; Add in offset to ringbase.
12 11 0773 2095 BRB AFTER_RINGBASE ; Rejoin common code.
0775 2096
0775 2097 RINGBASE_UV1:
0775 2098
51 0084 C5 D0 0775 2099 MOVL UCBSL_PDT(R5),R1 ; Get the PDT address.
51 00000208 8F C0 077A 2100 ADDL #PDT$RINGBASE,R1 ; Point to the ring base.
FF50 30 0781 2101 BSBW VIRT_TO_PHYAD ; Convert VA to physical address.
50 8ED0 0784 2102 POPL R0 ; Get physical address of ring base.\
0787 2103
0787 2104 AFTER_RINGBASE:
0787 2105
0787 2106 CPUDISP <<780,PURGE_780>,-
0787 2107 <750,PURGE_750>,-
0787 2108 <730,PURGE_730>,-
0787 2109 <790,PURGE_790>,-
0787 2110 <UV1,PURGE_UV1>,-
0787 2111 <8SS,PURGE_8SS>,-
0787 2112 >
07A1 2113
07A1 2114 PURGE_780:
07A1 2115 PURGE_790:
50 01 A8 07A1 2116 BISW #HS2_M_P1,R0 ; 11/790 acts same as 11/780.
07A4 2117 ; Ask for purge interrupts when
07A4 2118 PURGE_750: ; running on a VAX-11/780.
07A4 2119 PURGE_730:
07A4 2120 PURGE_UV1:
07A4 2121 PURGE_8SS:
07A4 2122
00B2 C5 50 B0 07A4 2123 MOVW R0,UCBSW_HOSTSTEP2(R5) ; Save communication area address and
07A9 2124 ; purge interrupt request bit in UCB.
07A9 2125
07A9 2126 ASSUME UCBSW_HOSTSTEP3 EQ UCBSW_PORTSTEP3+2
07A9 2127 MOVL R0,UCBSW_PORTSTEP3(R5) ; Trash UCBSW_PORTSTEP3 as we store the
07AE 2128 ; high word of R0 in UCBSW_HOSTSTEP3.
```

+ INIT_INIT_STEPS

```

07AE 2129
07AE 2130 ; Here we pickup the SYSGEN parameter UDABURSTRATE.
07AE 2131
50 00000000'GF 9A 07AE 2132 MOVZBL G^SCS$GB UDABURST,R0 ; R0 contains the burst rate setting.
50 50 02 78 07B5 2133 ASHL #HS4_V_BURST,R0,R0 ; Shift burst rate into position.
A9 07B9 2134 BISW3 #HS4_M_GO!- ; Set GO bit and last fail packet bits,
07BA 2135 HS4_M_LF,- ; or in burst rate,
00BA C5 50 03 07BA 2136 R0,- ; and save as HOST response to STEP 4.
07BF 2137 UCB$W_HOSTSTEP4(R5)
50 01 D0 07BF 2138
05 07BF 2139 MOVL S^#SS$_NORMAL,R0 ; Indicate success and
07C2 2140 RSB ; return to caller.
07C3 2141 .dsabl lsb

```


+ Hardware Initialization

```
07C3 2143      .SBTTL +      Hardware Initialization
07C3 2144
07C3 2145      :+
07C3 2146      : HARDWARE_INIT - internal subroutine
07C3 2147      :
07C3 2148      : Inputs:
07C3 2149      :      R5                      -Addr of UCB
07C3 2150      :
07C3 2151      : Outputs:
07C3 2152      :
07C3 2153      : -
07C3 2154
07C3 2155      CNTRLTYP_ARRAY:
0000 07C3 2156      .WORD   UDA50_CNTRLTYP
0001 07C5 2157      .WORD   LESI_CNTRLTYP
0005 07C7 2158      .WORD   TU81_CNTRLTYP
0006 07C9 2159      .WORD   UDA50A_CNTRLTYP
0007 07CB 2160      .WORD   RDRX_CNTRLTYP
0003 07CD 2161      .WORD   MAYA_CNTRLTYP
000D 07CF 2162      .WORD   QDA50_CNTRLTYP
FFFF 07D1 2163      .WORD   -1                      ; End of array fence.
FFFF 07D3 2164      .WORD   -1,-1,-1,-1,-1          ; Space for expansion.
07DD 2165      DEVTYP_ARRAY:
03 07DD 2166      .BYTE   DTS_UDA50
05 07DE 2167      .BYTE   DTS_LESI
06 07DF 2168      .BYTE   DTS_TU81P
04 07E0 2169      .BYTE   DTS_UDA50A
07 07E1 2170      .BYTE   DTS_RDRX
08 07E2 2171      .BYTE   DTS_TK50P
0B 07E3 2172      .BYTE   DTS_QDA50
00 00 00 00 00 00 07E4 2173      .BYTE   0,0,0,0,0,0      ; Space for expansion.
07EA 2174      SERVERS_ARRAY:
05 07EA 2175      .BYTE   <1aDISK_CONID>|<1aDUP_CONID>      ; UDA supports disks and DUP.
05 07EB 2176      .BYTE   <1aDISK_CONID>|<1aDUP_CONID>      ; Aztec supports disks and DUP.
06 07EC 2177      .BYTE   <1aTAPE_CONID>|<1aDUP_CONID>      ; TU81 supports tapes and DUP.
05 07ED 2178      .BYTE   <1aDISK_CONID>|<1aDUP_CONID>      ; UDA50A supports disks and DUP.
05 07EE 2179      .BYTE   <1aDISK_CONID>|<1aDUP_CONID>      ; RD/RX supports disks and DUP.
06 07EF 2180      .BYTE   <1aTAPE_CONID>|<1aDUP_CONID>      ; MAYA supports tapes and DUP.
05 07F0 2181      .BYTE   <1aDISK_CONID>|<1aDUP_CONID>      ; QDA50 supports disks and DUP.
00 00 00 00 00 00 07F1 2182      .BYTE   0,0,0,0,0,0      ; Space for expansion.
07F7 2183      UCODE_VER_ARRAY:
00 07F7 2184      .BYTE   0                      ; Minimum ucode version levels
07F8 2185      .BYTE   0                      ; UDA50A (0 since special purpose
00 07F8 2186      .BYTE   0                      ;   code already checks).
00 07F9 2187      .BYTE   0                      ; RC25 (0 for now)
00 07FA 2188      .BYTE   0                      ; TU81 (0 for now)
07FB 2189      .BYTE   0                      ; UDA50A (0 since special purpose
09 07FB 2190      .BYTE   9                      ;   code already checks).
00 07FC 2191      .BYTE   0                      ; RDRX.
00 07FD 2192      .BYTE   0                      ; MAYA.
00 07FE 2193      .BYTE   0                      ; QDA.
00 00 00 00 00 00 07FE 2193      .BYTE   0,0,0,0,0,0      ; Space for expansion.
0804 2194      OPC_MSG_ARRAY:
57 0804 2195      .BYTE   MSG$_UDA50MVER          ; UDA50 Opcom message number
5D 0805 2196      .BYTE   MSG$_RC25MVER          ; RC25 Opcom message number
5F 0806 2197      .BYTE   MSG$_TU81MVER          ; TU81 Opcom message number
57 0807 2198      .BYTE   MSG$_UDA50MVER          ; UDA50A Opcom message number
5E 0808 2199      .BYTE   MSG$_RDRXMVER          ; RDRX Opcom message number
```

+ Hardware Initialization

```

      5F 0809 2200      .BYTE MSG$_TU81MVER      ; Use TU81 Opcom message number
      080A 2201      ; for MAYA for now.
      57 080A 2202      .BYTE MSG$_UDA50MVER      ; Use UDA50 Opcom message number
      080B 2203      ; for QDA for now.
00 00 00 00 00 00 080B 2204      .BYTE 0,0,0,0,0,0 ; Space for expansion.
      0811 2205      .enabl lsb
      0811 2206      HARDWARE_INIT:
      0811 2207
      009C C5 8ED0 0811 2208      POPL UCB$_DPC(R5)      ; Remember return
      00A5 C5 97 0816 2209      DECB UCB$_INITCNT(R5)      ; Count # of consecutive times thru here.
      2A 12 081A 2210      BNEQ 8$      ; NEQ means that we have not exhausted
      081C 2211      ; consecutive retries.
      53 24 A5 D0 081C 2212      MOVL UCB$_CRB(R5),R3      ; R3 => CRB.
1C A3 2F AF 9E 0820 2213      MOVAB B^4$,CRB$_TOUTROUT(R3) ; Establish wakeup routine.
      0A C1 0825 2214      ADDL3 #INIT_DELTA,-          ; Establish a small delay.
      00000000 GF 0827 2215      G^EXE$GL ABSTIM,-
      18 A3 082C 2216      CRB$_DUETIME(R3)
      05 082E 2217      RSB      ; Evaporate for awhile.
      082F 2218 4$:
      082F 2219      SETIPL #IPL$_SCS      ; Lower IPL after wakeup.
      54 10 A3 D0 0832 2220      MOVL CRB$_AUXSTRUC(R3),R4 ; R4 => PDT.
      55 00DC C4 D0 0836 2221      MOVL PDT$_UCB0(R4),R5 ; R5 => UCB.
      F8BA CF 9E 083B 2222      MOVAB NULL_ROUTINE,-      ; Restabl. null wakeup routine.
      1C A3 083F 2223      CRB$_TOUTROUT(R3)
      05 90 0841 2224      MOVAB #NO_CONSEC_INITS,-      ; Reinit count that went to zero.
      00A5 C5 0843 2225      UCB$_INITCNT(R5)
      0846 2226 8$:
      04 A8 0846 2227      BISW #UCB$_PU_HRDINI,-      ; Set bit signalling
      68 A5 0848 2228      UCB$_DEVSTS(R5)      ; that we initing the port.
      54 0084 C5 D0 084A 2229      MOVL UCB$_PDT(R5),R4 ; R4 => PDT.
      53 0100 C4 D0 084F 2230      MOVL PDT$_PU_CSR(R4),R3 ; R3 => UDA CSR.
      0203 C4 94 0854 2231      CLRB PDT$_PURGEDP(R4) ; Prevent spurious purges.
      0858 2232 10$:
      0858 2233
      63 B4 0858 2234      CLRW UDAIP(R3)      ; Start hard init.
      085A 2235
      085A 2236 ; At least 100 micro seconds must pass between here
      085A 2237
      085A 2238      ASSUME UCB$_BOFF EQ UCB$_SVAPTE+4
      085A 2239      ASSUME UCB$_BCNT EQ UCB$_BOFF+2
      085A 2240      ASSUME UCB$_PU_BOFF EQ UCB$_PU_SVAPTE+4
      085A 2241      ASSUME UCB$_PU_BCNT EQ UCB$_PU_BOFF+2
      085A 2242
      085A 2243
      085A 2244      CPUDISP <<780,LOADUBA_COMMON>,-
      085A 2245      <750,LOADUBA_COMMON>,-
      085A 2246      <730,LOADUBA_COMMON>,-
      085A 2247      <790,LOADUBA_COMMON>,-
      085A 2248      <UV1,NOLOADUBA_UV1>,-
      085A 2249      <8SS,LOADUBA_8SS>,-
      085A 2250      >
      0874 2251
      50 00E0 C4 D0 0874 2252      LOADUBA_8SS:
      01 B1 0874 2253      MOVL PDT$_ADP(R4),R0      ; R0 => ADP.
      0E A0 0879 2254      CMPW #AT$_UBA,-          ; See if on UNIBUS adapter.
      04 13 087B 2255      ADPSQ ADPTYPE(R0)
      087D 2256      BEQL LOADUBA_COMMON      ; EQL implies UNIBUS adapter.
```


+ Hardware Initialization

```

087F 2257          BUG_CHECK          UDAPORT,FATAL      ; For now, if on BDA.
0883 2258
0883 2259 LOADUBA_COMMON:
0883 2260
00BC C5 7D 0883 2261          MOVQ      UCB$$_PU_SVAPTE(R5),- ; Copy parameters that allow mapping
78 A5      0887 2262          UCB$$_SVAPTE(R5) ; of communication area and buffers.
0889 2263          LOADUBA ; Load UNIBUS map registers with values
088F 2264          ; that map communication area and buffers.
088F 2265 NOLOADUBA_UV1:
088F 2266
088F 2267          DSBINT
0895 2268          WFIKPCH 15$,#2 ; Cause timeout to waste time.
089F 2269 15$:
089F 2270          SETIPL  UCB$$_FIPL(R5) ; Lower IPL after timeout.
0282 30 08A3 2271          BSBW      INIT_PU_PDT ; Initialize variable PDT data.
00AC C5 B4 08A6 2272          CLRW     UCB$$_PORTSTEP1(R5) ; Clear port responses so that
00B0 C5 B4 08AA 2273          CLRW     UCB$$_PORTSTEP2(R5) ; the error log record we create
00B4 C5 B4 08AE 2274          CLRW     UCB$$_PORTSTEP3(R5) ; will contain only the responses
00B8 C5 B4 08B2 2275          CLRW     UCB$$_PORTSTEP4(R5) ; received in this INIT attempt.
00A6 C5 B6 08B6 2276          INCW     UCB$$_NUMBINITS(R5) ; Increment # of times we have attempted
08BA 2277          ; to init port.
08BA 2278 ; and here
08BA 2279
02 A3 B0 08BA 2280          MOVW     UDASA(R3),- ; Read SA register.
00AC C5 B4 08BD 2281          UCB$$_PORTSTEP1(R5)
03 00AC C5 E1 08C0 2282          BBC      #PS1_V_ER,- ; Test for error condition
08C2 2283          UCB$$_PORTSTEP1(R5),30$ ; and branch around if clear.
01E9 31 08C6 2284 20$:
08C6 2285          BRW      HARD_RETRY ; For now only.
08C9 2286 30$:
08C9 2287          BBC      #PS1_V_S1,- ; Make sure we are in STEP1
F7 00AC C5 E1 08CB 2288          UCB$$_PORTSTEP1(R5),20$ ; or else go back.
08CF 2289
08CF 2290 ; In Step 1.
08CF 2291
08CF 2292          DSBINT
03 05 E5 08D5 2293          BBCC     #UCB$$_POWER,- ; Prepare to respond to STEP1.
03 64 A5 08D7 2294          UCB$$_STS(R5),35$ ; Power failure negates what we have
01D2 31 08DA 2295          BRW      HARDPower ; done until now.
08DD 2296 35$:
08DD 2297          MOVW     UCB$$_HOSTSTEP1(R5),- ; Else move value to SA register.
00AE C5 B0 08E1 2298          UDASA(R3)
02 A3      08E3 2299          WFIKPCH STEP1_TIMEOUT,#STEP1_LIMIT ; Wait for interrupt.
08ED 2300
08ED 2301          IOFORK ; Lower IPL and continue.
01FE 30 08F3 2302          BSBW     WAIT100US ; Call to wait for at least 100 uSecs.
02 A3 B0 08F6 2303          MOVW     UDASA(R3),- ; For slow TU81, copy UDASA after fork-
00AA C5 B4 08F9 2304          UCB$$_UDASA(R5) ; ing so as to give it enough time.
00AA C5 B0 08FC 2305          MOVW     UCB$$_UDASA(R5),- ; Save Port STEP2 start value
00B0 C5      0900 2306          UCB$$_PORTSTEP2(R5)
0903 2307
0903 2308 ; In Step 2.
0903 2309
06 00B0 C5 0F E0 0903 2310          BBS      #PS2_V_ER,UCB$$_PORTSTEP2(R5),37$ ; For now!!!!
0909 2311
03 00B0 C5 0C E0 0909 2312          BBS      #PS2_V_S2,- ; Make sure we are in STEP2.
090B 2313          UCB$$_PORTSTEP2(R5),40$

```


+ Hardware Initialization

```
01A0 31 090F 2314 37$: BRW HARD_RETRY ; Else branch to retry.
00B0 C5 91 090F 2315 40$: CMPB UCBSW_PORTSTEP2(R5),- ; Test whether controller echoed
00AF C5 0912 2316 40$: UCBSW_HOSTSTEP1+1(R5) ; fields correctly.
03 13 0916 2317 40$: BEQL 50$ ; EQL implies yes.
0194 31 0919 2318 40$: BRW HARD_RETRY ; Else branch to retry.
091E 2320 50$: DSBINT ; Prepare to respond to STEP2.
091E 2321 50$: BBCC #UCBSV POWER,- ; Power failure negates what we have
03 05 E5 0924 2322 50$: UCBSW_STS(R5),55$ ; done until now.
03 64 A5 0926 2323 50$: BRW HARDPower ; If power failure branch around.
0183 31 0929 2324 50$: MOVW UCBSW_HOSTSTEP2(R5),- ; Else move value to SA register.
00B2 C5 B0 092C 2325 50$: UDASA(R3)
02 A3 0930 2326 50$: WFIKPCB STEP2_TIMEOUT,#STEP2_LIMIT ; Wait for interrupt.
093C 2327 50$: IOFORK ; Lower IPL and continue.
01AF 30 0942 2328 50$: BSBW WAIT100US ; Call to wait for at least 100 uSecs.
02 A3 B0 0945 2329 50$: MOVW UDASA(R3),- ; For slow TU81, copy UDASA after fork-
00AA C5 0948 2330 50$: UCBSW_UDASA(R5) ; ing so as to give it enough time.
00AA C5 B0 094B 2331 50$: MOVW UCBSW_UDASA(R5),- ; Save Port STEP3 start value
00B4 C5 094F 2332 50$: UCBSW_PORTSTEP3(R5)
0952 2333 50$: ; In Step 3.
0952 2334 50$:
06 00B4 C5 0F E0 0952 2335 50$: BBS #PS3_V_ER,UCBSW_PORTSTEP3(R5),57$ ; For now!!!!
0958 2336 50$:
03 00B4 C5 0D E0 0958 2337 50$: BBS #PS3_V_S3,- ; Make sure we are in STEP3.
095A 2338 50$: UCBSW_PORTSTEP3(R5),60$
0151 31 095E 2339 50$: BRW HARD_RETRY ; Else branch to retry.
0961 2340 60$:
00B4 C5 91 0961 2341 60$: CMPB UCBSW_PORTSTEP3(R5),- ; Test whether controller echoed
00AE C5 0965 2342 60$: UCBSW_HOSTSTEP1(R5) ; fields correctly.
03 13 0968 2343 60$: BEQL 70$ ; EQL implies yes.
0145 31 096A 2344 60$: BRW HARD_RETRY ; Else branch to retry.
096D 2345 70$: DSBINT ; Prepare to respond to STEP3.
096D 2346 70$: BBCC #UCBSV POWER,- ; Power failure negates what we have
03 05 E5 0973 2347 70$: UCBSW_STS(R5),72$ ; done until now. No failure, continue.
03 64 A5 0975 2348 70$: BRW HARDPower ; So branch to HARDPower if failure.
0134 31 0978 2349 70$: MOVW UCBSW_HOSTSTEP3(R5),- ; Else move value to SA register.
00B6 C5 B0 097B 2350 70$: UDASA(R3)
02 A3 097F 2351 70$: WFIKPCB STEP3_TIMEOUT,#STEP3_LIMIT ; Wait for interrupt.
0981 2352 70$: IOFORK ; Lower IPL and continue.
098B 2353 70$: BSBW WAIT100US ; Call to wait for at least 100 uSecs.
0160 30 0991 2354 70$: MOVW UDASA(R3),- ; For slow TU81, copy UDASA after fork-
02 A3 B0 0994 2355 70$: UCBSW_UDASA(R5) ; ing so as to give it enough time.
00AA C5 0997 2356 70$: MOVW UCBSW_UDASA(R5),- ; Save Port STEP4 start value
00AA C5 B0 099A 2357 70$: UCBSW_PORTSTEP4(R5)
00B8 C5 099E 2358 70$: ; In Step 4.
09A1 2359 70$:
06 00B8 C5 0F E0 09A1 2360 70$: BBS #PS4_V_ER,UCBSW_PORTSTEP4(R5),77$ ; For now!!!!
09A1 2361 70$:
09A1 2362 70$:
09A1 2363 70$:
09A1 2364 70$:
09A1 2365 70$:
09A1 2366 70$:
09A1 2367 70$:
09A1 2368 70$:
09A1 2369 70$:
09A1 2370 70$:
```


+ Hardware Initialization

```
03 00B8 0E E0 09A7 2371 BBS #PS4_V S4,- ; Make sure we are in STEP4.
C5 09A7 2372 UCB$Q_PORTSTEP4(R5),80$
0102 31 09A9 2373 77$: BRW HARD_RETRY ; Else branch around to retry.
09AD 2374 80$: CMPZV #PS4_V_CNTRLTYP,- ; Here we assure that the controller
09B0 2375 09B2 2378 #PS4-S_CNTRLTYP,- ; microcode is upto rev level. First
00B8 C5 09B3 2379 UCB$Q_PORTSTEP4(R5),- ; see if we have a UDA50.
00 09B6 2380 #UDA50_CNTRLTYP
13 12 09B7 2381 BNEQ 90$ ; If NOT, branch around.
03 90 09B9 2382 MOVW #DT$ UDA50,- ; Fill in UCB devtype field with value
41 A5 09BB 2383 UCB$B_DEVTYPE(R5) ; for UDA50.
00 ED 09BD 2384 CMPZV #PS4_V_UCODEVER,- ; If a UDA50, see if microcode
04 09BF 2385 #PS4-S_UCODEVER,- ; upto rev by comparing against out of
00B8 C5 09C0 2386 UCB$Q_PORTSTEP4(R5),- ; date microcode version #.
01 09C3 2387 #1 ; Out of date version number.
06 12 09C4 2388 BNEQ 90$ ; NEQ implies Ucode is OK.
0116 31 09C6 2389 BRW UDA_OUTOFREV ; EQL implies inoperative Ucode.
09C9 2390
00E3 31 09C9 2391 GOTO_HARDP: BRW HARDPower ; Branch to faraway label.
09CC 2392 90$: BSBW STOCK_RSPRING ; Stock response ring after controller
01DB 30 09CC 2394 ; clears it.
09CF 2395
09CF 2396 DSBINT ; Prepare to respond to STEP4.
05 E4 09D5 2397 BBSC #UCB$V POWER,- ; Power failure negates what we have
EF 64 A5 09D7 2399 UCB$W_STS(R5),GOTO_HARDP; done until now.
00BA C5 B0 09DA 2400 MOVW UCB$W_HOSTSTEP4(R5),- ; Else move value to SA register.
02 A3 09DE 2401 UDASA(R3)
04 AA 09E0 2402 ENBINT ; Lower IPL and
68 A5 09E3 2403 BICW #UCB$M PU_HRDINI,- ; Reset bit signalling
01 B0 09E5 2404 UCB$W_DEVSTS(R5) ; that we initing the port.
00A8 C5 09E7 2405 MOVW #INIT_ATTNCODE,- ; Indicate what kind of error log record
00000000'GF 16 09E9 2406 UCB$W_ATTNCODE(R5) ; we are about to create.
09EC 2407 JSB G^ERL$DEVICEATTN ; Call to create error log record of INIT.
09F2 2408
09F2 2409 ; Here we have initialized the port. If it is a UDA, permanently allocate
09F2 2410 ; a Buffered Datapath and bias the usage count so that it is never released.
09F2 2411 ; This is done only on the first time thru here.
09F2 2412
03 05 E3 09F2 2413 BBCS #UCB$V PU_BDPATH,- ; Make sure first time thru.
68 A5 09F4 2414 UCB$W_DEVSTS(R5),140$
0095 31 09F7 2415 BRW 190$ ; If NOT first time, branch around.
09FA 2416 140$: EXTZV #PS4_V_CNTRLTYP,- ; Extract controller type and
04 EF 09FA 2417 #PS4-S_CNTRLTYP,- ; put it in on top of stack.
07 09FC 2418 UCB$Q_PORTSTEP4(R5),-(SP)
7E 00B8 C5 09FD 2419 CLRL R0 ; Clear loop counter.
50 D4 0A01 2420 150$:
FDBA CF40 6E B1 0A03 2422 CMPW (SP),CNTRLTYP_ARRAY[R0] ; Look for UQPORT type.
08 13 0A09 2423 BEQL 160$ ; EQL implies found.
FDB3 CF40 B5 0A0B 2424 TSTW CNTRLTYP_ARRAY[R0] ; See if at end of array.
30 19 0A10 2425 BLSS 170$ ; LSS implies NOT found.
50 D6 0A12 2426 INCL R0 ; Increment loop counter.
ED 11 0A14 2427 BRB 150$ ; And loop back.
```

+ Hardware Initialization

```
41 A5 FDC2 CF40 90 0A16 2428 160$:
54 0084 C5 D0 0A16 2429 MOV B DEVTYPE_ARRAY[R0],- ; Copy appropriate UQPORT type to
0128 C4 FDC3 CF40 90 0A1D 2430 UCB$B_DEVTYPE(R5) ; UCB.
0A1D 2431 MOVL UCB$L_PDT(R5),R4 ; R4 => PDT.
0A22 2432 MOV B SERVERS_ARRAY[R0],- ; Copy list of servers supported
0A2A 2433 PDT$B_SERVERS(R4) ; to PDT field.
51 FDC8 CF40 9A 0A2A 2434 MOVZBL UCODE_VER_ARRAY[R0],R1 ; Minimum acceptable ucode version
00 00 ED 0A30 2435 CMPZV #PS4_V_UCODEVER,- ; Compare our ucode version to
04 0A32 2436 #PS4_S_UCODEVER,- ; minimum acceptable level.
51 00B8 C5 0A33 2437 UCB$Q_PORTSTEP4(R5),R1
09 18 0A37 2438 BGEQ 170$ ; GEQ implies it IS acceptable.
51 FDC6 CF40 9A 0A39 2439 MOVZBL OPC_MSG_ARRAY[R0],R1 ; Opcom message number to R1.
00D1 30 0A3F 2440 BSBW SEND_OPC_MSG ; Subroutine to send message.
50 8ED0 0A42 2441 170$:
0A42 2442 POPL R0 ; Controller type to R0.
0A45 2443
00 50 D1 0A45 2444 CMPL R0,#UDA50_CNTRLTYP ; Is it UDA50?
05 13 0A48 2445 BEQL 180$ ; If YES, then go allocate.
06 50 D1 0A4A 2446 CMPL R0,#UDA50A_CNTRLTYP ; Is it UDA50A?
40 12 0A4D 2447 BNEQ 190$ ; If neither of above, no allocate.
0A4F 2448
0A4F 2449 180$:
0A4F 2450
0A4F 2451 ; Older UDA's on VAX-11/780 and VAX-11/790 systems have a purge problem
0A4F 2452 ; that corrupts data.
0A4F 2453
0A4F 2454 CPUDISP <<780,TESTUDA_780>,-
0A4F 2455 <750,UDA750>,-
0A4F 2456 <730,UDA730>,-
0A4F 2457 <790,TESTUDA_790>,-
0A4F 2458 <UV1,UDAUV1>,-
0A4F 2459 <8SS,UDA8SS>,-
0A4F 2460 >
0A69 2461 TESTUDA_780:
0A69 2462 TESTUDA_790:
00 ED 0A69 2463 CMPZV #PS4_V_UCODEVER,- ; If UDA on 780, see if microcode
04 0A6B 2464 #PS4_S_UCODEVER,- ; upto rev by comparing against out of
00B8 C5 0A6C 2465 UCB$Q_PORTSTEP4(R5),- ; date microcode version #.
02 0A6F 2466 #2 ; Out of date version number.
6D 15 0A70 2467 BLEQ UDA_OUTOFREV ; LEQ implies inoperative Ucode.
1B 11 0A72 2468 BRB 190$ ; Branch around allocation of perm. BDP.
0A74 2469 UDA750:
0A74 2470 UDA8SS:
00000000'GF 16 0A74 2471 JSB G^IOC$REQDATAP ; Permanently allocate a datapath.
50 24 A5 D0 0A7A 2472 MOVL UCB$L_CRB(R5),R0 ; R0 => CRB.
54 10 A0 D0 0A7E 2473 MOVL CRB$L_AUXSTRU0(R0),R4 ; R4 => PDT.
0A82 2474
37 A0 90 0A82 2475 MOV B CRB$L_INTD+VECSB_DATAPATH(R0),- ; Remember datapath permanently
0129 C4 0A85 2476 PDT$B_DATAPATH(R4) ; allocated in PDT.
37 A0 94 0A88 2477 CLRB CRB$L_INTD+VECSB_DATAPATH(R0) ; Clear CRB field.
0A8B 2478
012A C4 96 0A8B 2479 INCB PDT$B_BDPUSECNT(R4) ; Bias usage count to prevent dealloc.
0A8F 2480 UDA730:
0A8F 2481 UDAUV1:
0A8F 2482 190$:
0A8F 2483
0A8F 2484 ; Here we setup the CRB wakeup mechanism to periodically poll the SA register
```


+ Hardware Initialization

```
0A8F 2485 ; to determine if the U/QPORT has had an uncorrectable error.
0A8F 2486
50 24 A5 D0 0A8F 2487 MOVL UCBSL_CRB(R5),R0 ; R0 => CRB.
OF C1 0A93 2488 ADDL3 #SA_POLL_INTERVAL,- ; Establish SA polling interval.
00000000'GF 0A95 2489 G^EXESGL_ABSTIM,-
18 A0 0A9A 2490 CRBSL_DUE TIME(R0)
135B'CF 9E 0A9C 2491 MOVAB W^PUSSA_POLL,- ; Establish wakeup routine.
1C A0 0AA0 2492 CRBSL_TOUTROUT(R0)
0A8F 2493
50 01 D0 0AA2 2494 MOVL S^#SSS_NORMAL,R0 ; Return normal status.
0040 8F AA 0AA5 2495 200$: BICW #UCBSM_PU_INILOG,- ; Indicate Init Error Log no longer
68 A5 0AA9 2496 UCBSW_DEVSTS(R5) ; in progress (if it was).
009C D5 17 0AAB 2497 JMP @UCBSL_DPC(R5) ; return to caller.
0AAF 2498
0AAF 2500 HARDPOWER: ; We got a powerfailure while initing
0AAF 2501 ; the UDA. Simply start hardware init
0AAF 2502 ; over.
0AAF 2503 ENBINT ; Undo DSBINT.
0AB2 2504 HARD_RETRY:
0AB2 2505 MOVL UCBSL_PDT(R5),R4 ; R4 => PDT.
54 0084 C5 D0 0AB7 2506 MOVL PDTSL_PU_CSR(R4),R0 ; R0 => UDA CSR.
50 0100 C4 D0 0ABC 2507 MOVW UDASA(R0),UCBSW_UDASA(R5) ; Save error status in UCB for logging.
00AA C5 02 A0 B0 0AC2 2508 MOVW #FAIL_ATTNCODE,- ; Indicate what kind of error log record
00A8 C5 02 B0 0AC4 2509 UCBSW_ATTNCODE(R5) ; we are about to create.
06 06 E2 0AC7 2510 BBSS #UCBSV_PU_INILOG,- ; Indicate Init Error Log in progress,
00000000'GF 16 0ACC 2511 UCBSW_DEVSTS(R5),210$ ; and if in progress already, branch.
009C C5 DD 0AD2 2512 JSB G^ERL$DEVICEATTN ; Call to create error log record of INIT.
FD38 31 0AD6 2513 210$: PUSHL UCBSL_DPC(R5) ; Restore caller's return to stack.
0AD9 2514 BRW HARDWARE_INIT ; And branch back to restart hardware
0AD9 2515 ; init of UDA.
0AD9 2516
0AD9 2517 STEP1_TIMEOUT:
0AD9 2518 STEP2_TIMEOUT:
0AD9 2519 STEP3_TIMEOUT:
0AD9 2520
D3 11 0AD9 2521 SETIPL UCBSB_FIPL(R5) ; Lower IPL after timeout.
0ADD 2522 BRB HARD_RETRY ; Go to try again.
0ADF 2523
0ADF 2524 UDA_OUTOFREV:
0ADF 2525 .IF DF UDA50_BYPASS
0ADF 2526 BRB 90$ ; Bypass rejection.
0ADF 2527 .ENDC
0ADF 2528
00A8 C5 B0 0ADF 2529 MOVW #UCODE_ATTNCODE,- ; Indicate what kind of error log record
00000000'GF 16 0AE1 2530 UCBSW_ATTNCODE(R5) ; we are about to create.
0AEA 2531 JSB G^ERL$DEVICEATTN ; Call to create error log record of INIT.
51 57 8F 9A 0AEA 2532 MOVZBL #MSG$_UDA50MVER,R1 ; Message number to R1 for subroutine.
23 10 0AEE 2533 BSBB SEND_OPC_MSG ; Send message to OPCOM.
0AF0 2534
50 D4 0AF0 2535 CLRL R0 ; Indicate unable to init hardware.
B1 11 0AF2 2536 BRB 200$ ; Branch back to return.
0AF4 2537
0AF4 2538 .dsabl lsb
0AF4 2539
0AF4 2540
0AF4 2541 WAIT100US:
```

+ Hardware Initialization

```

05 0AF4 2542          TIMEDWAIT      TIME=#15      ; Wait for 150 uSecs.
    OB12 2543          RSB
    OB13 2544
    OB13 2545 ; R1 = MSG$_ message code.
    OB13 2546
    OB13 2547 SEND_OPC_MSG:
    OB13 2548
    53 54 3F BB OB13 2549          PUSHR    #^M<R0,R1,R2,R3,R4,R5> ; Save registers.
        51 D0 OB15 2550          MOVL     R1,R4 ; Message number to R4.
        00000000'GF 9E OB18 2551          MOVAB  G^SYS$GL OPRMBX,R3 ; R3 => operator's mailbox.
        00000000'GF 16 OB1F 2552          JSB   G^EXE$SNDEVMSG ; Call to send operator message.
        3F BA OB25 2553          POPR     #^M<R0,R1,R2,R3,R4,R5> ; Restore registers.
    05 OB27 2554          RSB ; Return to caller.

```


+ INIT_PU_PDT Fill in variable

```
0B28 2556      .SBTTL +      INIT_PU_PDT      Fill in variable
0B28 2557      .SBTTL +      PDT data
0B28 2558
0B28 2559 :+
0B28 2560 : INIT_PU_PDT - internal subroutine to initialize (or re-initialize)
0B28 2561 : variable PDT data. This routine assumes that BUILD_PDT has been called
0B28 2562 : prior to the activation of this routine.
0B28 2563
0B28 2564 : INIT_PU_PDT is called from HARDWARE_INIT, and initializes the Q headers
0B28 2565 : in the PDT and also initializes the Ring structures by filling the response
0B28 2566 : ring with available buffers and then placing all the rest of the buffers on
0B28 2567 : the free Q.
0B28 2568
0B28 2569 : Inputs:
0B28 2570
0B28 2571 :      R4                      -Addr of PDT
0B28 2572
0B28 2573 : Outputs:
0B28 2574
0B28 2575 :      R0-R2                    -Destroyed
0B28 2576 :      Other registers          -Preserved
0B28 2577
0B28 2578 :      Response ring filled
0B28 2579 :      Free Q filled
0B28 2580 :      PDT$$_PU_SNDQFL initialized
0B28 2581 :      PDT$$_PU_BUFQFL initialized
0B28 2582 : -
0B28 2583
0B28 2584 INIT_PU_PDT:
0B28 2585
0B28 2586      ASSUME  PDT$$_CMDINT      EQ      PDT$$_COMAREA+4
0B28 2587      ASSUME  PDT$$_RSPINT      EQ      PDT$$_CMDINT+2
0200 C4      7C 0B28 2588      CLRQ      PDT$$_COMAREA(R4)
0B2C 2589
0B2C 2590      ASSUME  PDT$$_CPOLLINX      EQ      PDT$$_CRINGINX+1
0B2C 2591      ASSUME  PDT$$_CRINGCNT      EQ      PDT$$_CPOLLINX+1
0B2C 2592      ASSUME  PDT$$_RRINGINX      EQ      PDT$$_CRINGCNT+1
0B2C 2593      ASSUME  PDT$$_RPOLLINX      EQ      PDT$$_RRINGINX+1
0B2C 2594      ASSUME  PDT$$_RPOOLLINX      EQ      PDT$$_RPOLLINX+1
0B2C 2595      ASSUME  PDT$$_NOCURCON      EQ      PDT$$_RRINGCNT+1
0B2C 2596      ASSUME  PDT$$_CONBITMAP      EQ      PDT$$_NOCURCON+1
0120 C4      7C 0B2C 2597      CLRQ      PDT$$_CRINGINX(R4)
0B30 2598
0B30 2599      MOVL      #UDAB$T TEXT,-      ; Save UDA port SCS datagram header
00B8 C4      D0 0B32 2600      PDT$$_DGOVRHD(R4)      ; size in PDT.
0B35 2601      MOVL      #UDAB$T TEXT,-      ; Save UDA port SCS message header
00B4 C4      D0 0B37 2602      PDT$$_MSGHDRSZ(R4)      ; size in PDT.
0B3A 2603
0B3A 2604 : Initialize PDT Q headers.
0B3A 2605
0B3A 2606      MOVAB      PDT$$_WAITQFL(R4),-      ; Empty wait queue header
00AC C4      9E 0B3E 2607      PDT$$_WAITQFL(R4)      ;
00AC C4      9E 0B41 2608      PDT$$_WAITQFL(R4),-      ; Empty wait queue header
00B0 C4      9E 0B45 2609      PDT$$_WAITQBL(R4)      ;
0B48 2610
0108 C4      9E 0B48 2611      MOVAB      PDT$$_PU_FQFL(R4),-      ; Free Q header.
0108 C4      9E 0B4C 2612      PDT$$_PU_FQFL(R4)
```

```
+ PDT data

0108 C4 9E 0B4F 2613 MOVAB PDT$$_PU_FQFL(R4),- ; Free Q header.
010C C4 0B53 2614 PDT$$_PU_FQBL(R4)
0108 C4 9E 0B56 2615 MOVAB PDT$$_PU_FQFL(R4),- ; Point Free Q pointer to Free Q. This
0104 C4 0B5A 2616 PDT$$_PU_FQPTR(R4) ; pointer is used to test for
0B5D 2617 ; emptiness of Free Q.
0B5D 2618
0110 C4 9E 0B5D 2619 MOVAB PDT$$_PU_SNDQFL(R4),- ; Send Q header.
0110 C4 0B61 2620 PDT$$_PU_SNDQFL(R4)
0110 C4 9E 0B64 2621 MOVAB PDT$$_PU_SNDQFL(R4),- ; Send Q header.
0114 C4 0B68 2622 PDT$$_PU_SNDQBL(R4)
0B6B 2623
0118 C4 9E 0B6B 2624 MOVAB PDT$$_PU_BUFQFL(R4),- ; Buffer wait Q header.
0118 C4 0B6F 2625 PDT$$_PU_BUFQFL(R4)
0118 C4 9E 0B72 2626 MOVAB PDT$$_PU_BUFQFL(R4),- ; Buffer wait Q header.
011C C4 0B76 2627 PDT$$_PU_BUFQBL(R4)
0B79 2628
0B79 2629 ; Initialize initial credits for all possible connections.
0B79 2630
00F4 C4 01 B0 0B79 2631 MOVW #1,PDT$$_PU_CRED0(R4) ; Initial credit of 1.
00F6 C4 01 B0 0B7E 2632 MOVW #1,PDT$$_PU_CRED1(R4) ; Initial credit of 1.
00F8 C4 01 B0 0B83 2633 MOVW #1,PDT$$_PU_CRED2(R4) ; Initial credit of 1.
00FA C4 01 B0 0B88 2634 MOVW #1,PDT$$_PU_CRD255(R4) ; Initial credit of 1.
0B8D 2635
0B8D 2636 ; Following loop is for arrays that are RINGSIZE long.
0B8D 2637
50 0F D0 0B8D 2638 MOVL #UDASK_RINGSIZE-1,R0 ; Initialize loop counter.
0B90 2639 10$:
012C C440 01 8E 0B90 2640 MNEGB #1,PDT$$_CRCONTENT(R4)[R0] ; Nothing in command slot.
013C C440 01 8E 0B96 2641 MNEGB #1,PDT$$_RRCONTENT(R4)[R0] ; Nothing in response slot.
0248 C440 D4 0B9C 2642 CLRL PDT$$_CMDRING(R4)[R0] ; Clear command ring slot.
0208 C440 D4 0BA1 2643 CLRL PDT$$_RSRING(R4)[R0] ; Clear response ring slot.
E7 50 F4 0BA6 2644 SOBGEQ R0,10$ ; Loop thru all array elements.
05 0BA9 2645 RSB ; Return to caller. Rest of PDT init
0BAA 2646 ; called explicitly at label below.
0BAA 2647
0BAA 2648 STOCK_RSPRING:
0BAA 2649
0BAA 2650 ; Loop thru all buffers (there are 2*RINGSIZE of them) and put them on the
0BAA 2651 ; response ring or the free Q.
0BAA 2652
50 D4 0BAA 2653 CLRL R0 ; Initialize loop variable.
0BAC 2654 10$:
52 014C C440 D0 0BAC 2655 MOVL PDT$$_BDTABLE(R4)[R0],R2 ; R2 => buffer[R0]
50 DD 0BB2 2656 PUSHL R0 ; Remember loop variable before call.
02E6 30 0BB4 2657 BSBW Q_DEALLOC_BUF ; Put buffer on response ring or free
0BB7 2658 ; Q, whichever is appropriate.
50 8ED0 0BB7 2659 POPL R0 ; Restore loop variable.
EE 50 20 F2 0BBA 2660 AOBLS #2*UDASK_RINGSIZE,R0,10$ ; Loop thru all buffers.
0BBE 2661
05 0BBE 2662 RSB ; And return to caller.
```



```

0BBF 2664 .SBTTL + BUILD_PB_SB Build System Block and Path Block
0BBF 2665
0BBF 2666 :+
0BBF 2667 BUILD_PB_SB - Build and fill in the System Block and the Path Block.
0BBF 2668
0BBF 2669 This portion of the UDA port driver is responsible for adding
0BBF 2670 the UDA to the system-wide configuration database. It is invoked
0BBF 2671 as a one time initialization routine.
0BBF 2672
0BBF 2673 The system wide configuration database consists of:
0BBF 2674
0BBF 2675
0BBF 2676 SCSSGQ_CONFIG
0BBF 2677 |
0BBF 2678 v
0BBF 2679 System Block ----> Path Block ----> Path Block ---->...
0BBF 2680 |
0BBF 2681 v
0BBF 2682 System Block ----> Path Block ---->...
0BBF 2683 |
0BBF 2684 v
0BBF 2685 ...
0BBF 2686
0BBF 2687 Only systems and paths with open port-port VC's are kept on the
0BBF 2688 above list.
0BBF 2689
0BBF 2690 For each UDA (AZTEC or TU81) we build a Path Block and a System Block,
0BBF 2691 initialize them and link them into the systemwide configuration database.
0BBF 2692
0BBF 2693 Inputs:
0BBF 2694
0BBF 2695 R4 -Addr of PDT
0BBF 2696 R5 -Addr of UCB
0BBF 2697
0BBF 2698 :-
0BBF 2699
0BBF 2700 BUILD_PB_SB:
0BBF 2701
0BBF 2702 MOVL #PBSK_LENGTH,R1 ; Get size of a pathblock
0BBF 2703 BSBW ALLOC_POOL ; Allocate one from pool
0BBF 2704 BLBS R0,10$ ; Branch around if success.
0BBF 2705 BRW PB_ALLOC_FAIL ; Branch if no pool
0BBF 2706 10$:
0BBF 2707 OBCF 2707 MOVL R2,R3 ; Set PB addr in stable register
0BBF 2708 OBD2 2708 MOVL R2,PB$L_FLINK(R2) ; This will be the only Path
0BBF 2709 OBD5 2709 ; Block on this list.
0BBF 2710 OBD5 2710 MOVL R2,PB$L_BLINK(R2) ; FLINK and BLINK point here.
0BBF 2711 OBD9 2711
0BBF 2712 OBD9 2712 MOVW R1,PB$W_SIZE(R2) ; Set structure size
0BBF 2713 OBDD 2713
0BBF 2714 OBDD 2714 ASSUME PBSB_SUBTYP EQ PBSB_TYPE+1
0BBF 2715 OBDD 2715 MOVW #DYN$C_SCS+<DYN$C_SCS_PBA8>,- ; Set struct type, subtype
0BBF 2716 OBE1 2716 PBSB_TYPE(R2)
0BBF 2717 OBE3 2717
0BBF 2718 OBE3 2718 MOVW #PB$C_OPEN,PB$W_STATE(R2) ; Always in OPEN state.
0BBF 2719 OBE7 2719
0BBF 2720 OBE7 2720 MOVL UCB$L_DDB(R5),R1 ; R1 => DDB for port device.

```

+ BUILD_PB_SB Build System Block and Pat

15 A1	D0	OBEB	2721	MOVL	DDBST_NAME+1(R1),-	; Copy port name to port block.		
24 A2		OBEE	2722		PBST_CPORT_NAME(R2)			
27 A2	30	90	OBFO	2723	MOVB	#A/0/,PBST_LPORT_NAME+3(R2)	; Unit is always 0.	
2C A2	54	D0	OBF4	2724				
			OBF4	2725	MOVL	R4,PB\$\$_PDT(R2)	; Save PDT pointer in Path Block.	
			OBFB	2726				
38 A2	9E		OBFB	2727	MOVAB	PB\$\$_WAITQFL(R2),-	; Initialize empty Q header.	
38 A2			OBFB	2728		PB\$\$_WAITQFL(R2)		
38 A2	9E		OBFD	2729	MOVAB	PB\$\$_WAITQFL(R2),-		
3C A2			OC00	2730		PB\$\$_WAITQBL(R2)		
			OC02	2731				
			OC02	2732				
			OC02	2733				
							; Now build System Block.	
51	00000060	8F	D0	OC02	2734	MOVL	#SB\$\$_LENGTH,R1	; Get size of SB
	0080		30	OC09	2735	BSBW	ALLOC_POOL	; Allocate from nonpaged pool
	59	50	E9	OC0C	2736	BLBC	R0,SB_ALLOC_FAIL	; Branch if no pool
30 A3	52	D0	OC0F	2737	MOVL	R2,PB\$\$_SBLINK(R3)	; Save System Block address in PB.	
00FC C4	52	D0	OC13	2738	MOVL	R2,PDT\$\$_PU_SB(R4)	; Save System Block address in PDT.	
			OC18	2739				
08 A2	51	B0	OC18	2740	MOVW	R1,SB\$\$_SIZE(R2)	; Set struct size	
0760	8F	B0	OC1C	2741	MOVW	#DYN\$\$_SCS+<DYN\$\$_SCS_SB\$8>,-	; Set structure type	
0A A2			OC20	2742		SB\$\$_TYPE(R2)	; and subtype	
OC A2	OC A2	9E	OC22	2743	MOVAB	SB\$\$_PBFL(R2),SB\$\$_PBFL(R2)	; Establish Q head of path	
10 A2	OC A2	9E	OC27	2744	MOVAB	SB\$\$_PBFL(R2),SB\$\$_PBBL(R2)	; blocks. And Q this path	
OC B2	63	OE	OC2C	2745	INSQUE	(R3),@SB\$\$_PBFL(R2)	; block to head of that Q.	
14 A2	53	D0	OC30	2746	MOVL	R3,SB\$\$_PB\$CONNX(R2)	; Let this path be Next path.	
			OC34	2747				
			OC34	2748				
			OC34	2749				
			OC34	2750				
			OC34	2751				
			OC34	2752				
			OC34	2753				
			OC34	2754				
			OC34	2755				
			OC34	2756				
50	24 A5	D0	OC34	2757	MOVL	UCB\$\$_CRB(R5),R0	; R0 => CRB.	
50	2C A0	D0	OC38	2758	MOVL	CRB\$\$_INTD+VEC\$\$_IDB(R0),R0	; R0 => IDB.	
	60	D0	OC3C	2759	MOVL	IDB\$\$_CSR(R0),-	; Move CSR to low longword of	
	18 A2		OC3E	2760		SB\$\$_SYSTEMID(R2)	; SYSTEMID.	
50	14 A0	D0	OC40	2761	MOVL	IDB\$\$_ADP(R0),R0	; R0 => ADP.	
	OC A0	B0	OC44	2762	MOVW	ADP\$\$_TR(R0),-	; Move nexus number to high	
	1C A2		OC47	2763		SB\$\$_SYSTEMID+4(R2)	; word of SYSTEMID.	
1D A2	80 8F	88	OC49	2764	BISB	#A\$X80,SB\$\$_SYSTEMID+5(R2)	; Set high bit on in SYSTEMID.	
			OC4E	2765				
00000000	'GF	7D	OC4E	2766	MOVQ	G^EXE\$\$_GQ SYSTIME,-	; Copy current time to boot time	
2C A2			OC54	2767		SB\$\$_SWINCARN(R2)	; of this port.	
5C A2		D4	OC56	2768	CLRL	SB\$\$_CSB(R2)	; Clear link to newest CSB.	
			OC59	2769				
			OC59	2770				
			OC59	2771				
							; Here Link System Block onto system list.	
50	00000000	'GF	DE	OC59	2772	MOVAL	G^SCS\$\$_GQ_CONFIG,R0	; R0 => Systemwide list head of system
			OC60	2773			blocks.	
04 B0	62	OE	OC60	2774	INSQUE	(R2),@4(R0)	; Queue to tail of list.	
50	01	D0	OC64	2775	MOVL	S^#SS\$\$_NORMAL,R0	; Indicate success and	
		05	OC67	2776	RSB		; return to caller.	
			OC68	2777				

PUDRIVER
V04-000

K 12

+ BUILD_PB_SB Build System Block and Pat 16-SEP-1984 01:05:05 VAX/VMS Macro V04-00
5-SEP-1984 00:17:10 [DRIVER.SRC]PUDRIVER.MAR;1

Page 60
(2)

50 D4 0C68 2778 PB_ALLOC_FAIL:
 0C68 2779 SB_ALLOC_FAIL:
 0C68 2780 CLRL R0
 0C6A 2781 RSB

; Indicate failure and
; return to caller.

```

+ UPDATE_PB_SB
      OC6B 2783      .SBTTL +      UPDATE_PB_SB
      OC6B 2784
      OC6B 2785      ;+
      OC6B 2786      ; UPDATE_PB_SB - routine to update fields in the PB (and potentially the SB)
      OC6B 2787      ; after the data has been obtained.
      OC6B 2788      ;
      OC6B 2789      ; Inputs:
      OC6B 2790      ; R5 => UCB
      OC6B 2791
      OC6B 2792      UPDATE_PB_SB:
      OC6B 2793
50      0084 C5      D0      OC6B 2794      MOVL      UCB$PDT(R5),R0      ; R0 => PDT.
50      00FC C0      D0      OC70 2795      MOVL      PDT$PU_SB(R0),R0      ; R0 => SB.
      14      13      OC75 2796      BEQL      10$      ; EQL means we failed to alloc SB.
50      14 A0      D0      OC77 2797      MOVL      SB$PBCONNX(R0),R0      ; R0 => PB.
      08      EF      OC7B 2798      EXTZV      #PS2_V_PORTTYPE,-      ; Extract Port Type returned by hardware
      03      OC7D 2800      #PS2_S_PORTTYPE,-      ; initialization of controller.
      00B0 C5      OC7E 2801      UCB$PORTSTEP2(R5),-
      14 A0      OC81 2802      PB$RPORT_TYP(R0)
      00      EF      OC83 2803
      00      OC83 2804      EXTZV      #PS4_V_UCODEVER,-      ; Extract ucode version returned by
      00      OC85 2805      #PS4_S_UCODEVER+,-      ; hardware init of controller.
      00B8 C5      OC85 2806      PS4_S_CNTRLTYP,-
      18 A0      OC85 2807      UCB$PORTSTEP4(R5),-
      OC89 2808      PB$RPORT_REV(R0)
      05      OC8B 2809      10$:
      OC8B 2810      RSB      ; Return to caller.

```


+ ALLOC_POOL

```
.SBTTL +      ALLOC_POOL
```

0C8C	2812
0C8C	2813
0C8C	2814
0C8C	2815
0C8C	2816
0C8C	2817
0C8C	2818
0C8C	2819
0C8C	2820
0C8C	2821
0C8C	2822
0C8C	2823
0C8C	2824
0C8C	2825
0C8C	2826
0C8C	2827
0C8C	2828
0C8C	2829
0C8C	2830

```

;+
; This subroutine allocates and zeroes nonpaged pool.
; It is assumed here that the call here is always made from a called
; internal subroutine, implying that two return points must be popped
; off the stack if the thread is suspended.

```

Inputs:R1
R5

```

-# bytes of pool to allocate
-Addr of UCB

```

Outputs:R0
R1
R2

```
-0/1 for fail/success
-# bytes actually allocated
-Addr of buffer allocated
```

ALLOC_POOL:

; Allocate and zero pool

	009C	C5	8ED0
		53	DD
	00000000	'GF	16
	11	50	E9
		37	BB
00	6E	00	2C
		37	BA
		53	8ED0
	009C	D5	17
00A0	C5	51	D0
		10	A5
	14	A5	54
		0C	A5
			8ED0
	53	24	A5
1C	A3	D1	'AF
			01
	00000000	'GF	
		18	A3
			05
	55	10	A3
			10
		0A	A5
			05
55	00DC	C5	13
			D0
		F412	CF
		1C	A3
51	00A0	C5	D0
	53	10	A5
			7D

OC8C	2834
OC91	2835
OC93	2836
OC99	2837
OC9C	2838
OC9E	2839
OCA4	2840
OCA6	2841
OCA9	2842
OCAD	2843
OCAD	2844
OCAD	2845
OCAD	2846
OCB2	2847
OCB6	2848
OCBA	2849
OCBE	2850
OCBE	2851
OCCE	2852
OCCE	2853
OCCE	2854
OCCE	2855
OCDO	2856
OCDD	2857
OCDD	2858
OCDD	2859
OCDD	2860
OCDA	2861
OCDC	2862
OCDE	2863
OCE3	2864
OCE3	2865
OCE7	2866
OCE9	2867
OCEE	2868

POPL
PUSHL
JSB
BLBC
PUSHR
MOVC5
POPR
POPL
JMP

MOVL
POPL
MOVL
POPL

MOVL
MOVAB
ADDL3

RSB

SETIP
MOVL
CMPB

BEQL
MOVL

MOVAB

MOVL
MOVQ

```

UCB$$_DPC(R5)
R3
G^EXE$ALONONPAGED
R0,10$
#^M<R0,R1,R2,R4,R5>
#0,(SP),#0,R1,(R2)
#^M<R0,R1,R2,R4,R5>
R3
@UCB$$_DPC(R5)

R1,UCB$$_PU_ALLOC(R5)
UCB$$_FR3(R5)
R4,UCB$$_FR4(R5)
UCB$$_FPC(R5)

UCB$$_CRB(R5),R3
B^20$,CRB$$_TOUTROUT(R3)
#ALLOC_DELTA,-
G^EXE$GL_ABSTIM,-
CRB$$_DUETIME(R3)

#IPL$$_SCS
CRB$$_AUXSTRUC(R3),R5
#DYN$$_UCB,-
UCB$$_TYPE(R5)
30$
PDT$$_UCB0(R5),R5

NULL_ROUTINE,-
CRB$$_TOUTROUT(R3)
UCB$$_PU_ALLOC(R5),R1
UCB$$_FR3(R5),R3

```

```

; Remember return
; Save R3.
; Allocate from nonpaged pool
; Skip clearing structure if failure
; Save MOVC registers
; Zero initialize structure
; Restore MOVC registers
; Restore R3.
; Return to caller.
; Here we suffered an allocation failure.
; Prepare to wait awhile before trying
; again.
; Save size of block to allocate.
; Pop given R3 into save area.
; Save R4.
; Save caller's caller's return.

; R3 => CRB.
) ; Establish wakeup routine.
; Establish a small delay.

; Evaporate for awhile.

; Lower IPL after wakeup.
; R5 => UCB or PDT.
; Determine which, UCB or PDT.

; EQL implies UCB.
; If PDT, go one level deeper so that
; here R5 => UCB.
; Restablish null wakeup routine.

; Restore R1 = size of block to alloc.
; Restore R3 and R4.

```

PUDRIVER
V04-000

+ ALLOC_POOL

OC	A5	DD	OCF2	2869
009C	C5	DD	OCF5	2870
	91	11	OCF9	2871

N 12

16-SEP-1984 01:05:05 VAX/VMS Macro V04-00 Page 63
5-SEP-1984 00:17:10 [DRIVER.SRC]PUDRIVER.MAR;1 (2)

PUSHL UCB\$FPC(R5)
PUSHL UCB\$DPC(R5)
BRB ALLOC_POOL

; Re-establish caller's caller's return poin
; Re-establish caller's return point.
; Go try again.

UNIMPLEMENTED FORK PROCESS CALLS

```

OCFB 2873 .SBTTL UNIMPLEMENTED FORK PROCESS CALLS
OCFB 2874 FPC$ACCEPT::
OCFB 2875 FPC$ALLOCDG::
OCFB 2876 FPC$DEALLOCDG::
OCFB 2877 FPC$MAP::
OCFB 2878 FPC$MAPBYPASS::
OCFB 2879 FPC$MAPIRBPYP::
OCFB 2880 FPC$QUEUEMDGS::
OCFB 2881 FPC$REJECT::
OCFB 2882 FPC$REQDATA::
OCFB 2883 FPC$SENDDATA::
OCFB 2884 FPC$SENDDG::
OCFB 2885 FPC$SENDMSG::
OCFB 2886
OCFB 2887 FPC$READCOUNT::
OCFB 2888 FPC$RLSCOUNT::
OCFB 2889 FPC$MAINTFCN::
OCFB 2890 FPC$SENDRGDG::
OCFB 2891 FPC$STOP_VCS::
OCFB 2892 BUG_CHECK

```

UDAPORT,FATAL

MRESET and MSTART

```
OCFF 2894      .SBTTL MRESET and MSTART
OCFF 2895
OCFF 2896      :+
OCFF 2897      : FPC$MRESET - causes connection ERROR ROUTINE to be called for all open
OCFF 2898      : connections and then does a HARDWARE INIT of the port.
OCFF 2899      : FPC$MSTART is a NOP.
OCFF 2900
OCFF 2901      : Inputs:
OCFF 2902      : R4 => PDT
OCFF 2903
OCFF 2904      : Outputs:
OCFF 2905      : All registers preserved, however caller is returned to before the
OCFF 2906      : thread started by FPC$MRESET finishes.
OCFF 2907      :-
OCFF 2908
OCFF 2909      FPC$MRESET::
OCFF 2910
55      3F      BB      OCFF 2911      PUSHR      #^M<R0,R1,R2,R3,R4,R5>      : Save caller's registers.
      OODC C4      DO      OD01 2912      MOVL      PDT$UCB0(R4),R5      : R5 => UCB for port.
      F5BA      30      OD06 2913      BSBW      POST_POWER_FORK      : Start thread to call error routines
      OD09 2914      : and to do hardware init.
      3F      BA      OD09 2915      POPR      #^M<R0,R1,R2,R3,R4,R5>      : Restore caller's registers.
      OD0B 2916      FPC$MSTART::
      05      OD0B 2917      RSB      : Return to caller.
```


CONNECTION MANAGEMENT CALLS

```
ODOC 2919      .SBTTL CONNECTION MANAGEMENT CALLS
ODOC 2920      .SBTTL +      FPC$CONNECT,      COMPLETE PROCESSING A CONNECT
ODOC 2921
ODOC 2922      :+
ODOC 2923      : This routine is JMP'ed to from SCSS$CONNECT with a CDT allocated
ODOC 2924      : (and in the closed state) and initialized with the SYSAP's
ODOC 2925      : connect parameters or 0's for fields not yet used. FPC$CONNECT
ODOC 2926      : does port-specific processing.
ODOC 2927
ODOC 2928      : The UDA port driver only supports one connection at a time. The first
ODOC 2929      : thing that FPC$CONNECT checks for is that there are no current connections
ODOC 2930      : on this port. Then if there are none we check that the target of the
ODOC 2931      : connection is one of the ones supported. If so, then the initial credits
ODOC 2932      : allotted to this connection are granted, the connection ID is saved, the
ODOC 2933      : CDT address is stored and the CDT state is set to open.
ODOC 2934
ODOC 2935      Inputs:
ODOC 2936
ODOC 2937      R3      -Addr of CDT
ODOC 2938      R4      -Addr of PDT
ODOC 2939
ODOC 2940      CDT initialized as follows:
ODOC 2941
ODOC 2942      CDT$L_LCONID      -Local conid
ODOC 2943      MSGINPUT      -Addr to call in SYSAP for rec'd msgs
ODOC 2944      DGINPUT      -Addr to call in SYSAP for rec'd dgs
ODOC 2945      ERRADDR      -Addr to call in SYSAP for connection errors
ODOC 2946      RSTATION      -Remote station addr
ODOC 2947      PDT      -Addr of PDT
ODOC 2948      MINSEND      -Minimum send credit req'd by SYSAP
ODOC 2949      INITLREC      -Initial credit extended by SYSAP
ODOC 2950      DGREC      -Initial # of dg's queued
ODOC 2951      STATE      -CLOSED
ODOC 2952      PB      -Addr of selected PB to remote system
ODOC 2953      WAITQFL/BL      -Set to show no entries
ODOC 2954      RPROCNAM      -Addr of dest process name
ODOC 2955      LPROCNAM      -Addr of local process name
ODOC 2956      CONDAT      -Addr of connect data
ODOC 2957
ODOC 2958      other CDT fields      -0
ODOC 2959
ODOC 2960
ODOC 2961      (SP)      -Return PC in SYSAP
ODOC 2962
ODOC 2963      Outputs:
ODOC 2964
ODOC 2965      R0      -Status: SSS_NORMAL, SSS_FAILRSP,
ODOC 2966      SSS_REJECT, SSS_INSMEM
ODOC 2967      R1      -Reject reason or fail response reason
ODOC 2968      if R0 = REJECT or FAILRSP
ODOC 2969      R2      -Addr of ACCEPT_REQ if R0 = success
ODOC 2970      other registers      -Preserved
ODOC 2971
ODOC 2972      CDT$L_RCONID - UDA virtual circuit number
ODOC 2973      :-
ODOC 2974
ODOC 2975      .ENABL LSB
```

```
+ FPC$CONNECT, COMPLETE PROCESSING A CON

4B 53 49 44 24 50 43 53 4D 0D0C 2976 VCONAM: .ASCII /MSCP$DISK/      ; Name for UDA VC 0
                                00000009 0D15 2977 VCONAMLEN = -VCONAM      ; name length
45 50 41 54 24 50 43 53 4D 0D15 2978 VC1NAM: .ASCII /MSCP$TAPE/      ; Name for UDA VC 1
                                00000009 0D1E 2979 VC1NAMLEN = -VC1NAM      ; name length
                                50 55 44 0D1E 2980 VC2NAM: .ASCII /DUP/      ; Name for UDA VC 2
                                00000003 0D21 2981 VC2NAMLEN = -VC2NAM      ; name length
                                0D21 2982
                                0D21 2983 FPC$CONNECT::
                                0D21 2984
50 00DC C4 D0 0D21 2985        MOVL   PDT$L_UCB0(R4),R0      ; R0 => UCB.
    04 E1 0D26 2986        BBC      #UCB$V_ONLINE,-          ; Reject connect if offline.
    38 64 A0 0D28 2987        UCB$W_STS(R0),CON_NO_NODE
                                0D28 2988
                                0D28 2989        $CHK_CDTSTATE -      ; Verify that CDT state
                                0D28 2990        CLOSED,-          ; is closed; if not,
                                0D28 2991        ERROR=STATE_ERR      ; caller made error
                                0D34 2992
                                0D34 2993        ; If the desired process name is MSCP$DISK, MSCP$TAPE or DUP, then the
                                0D34 2994        ; connection can be completed.
                                0D34 2995
                                0D34 2996        PUSHL   R3          ; Remember R3=>CDT.
                                0D36 2997        BBS      #UDASV_STOPPED,-      ; Only allow Connections to DUP if
                                0D38 2998        UCB$B_UDAFLAGS(R0),-      ; Port is closed.
                                0D3B 2999        TSTVC2
20 CC AF 09 2D 0D3C 3000        CMPC5   #VCONAMLEN,VCONAM,#^A/ /,-
50 B3 10 0D41 3001        #16,@CDT$L_RPROCNAME(R3)      ; Is the connection to MSCP$DISK?
    5F 13 0D44 3002        BEQL      CONVCO      ; If yes make connection to VC0
                                0D46 3003
                                0D46 3004        MOVL   (SP),R3      ; Refresh R3 => CDT.
20 C8 AF 09 2D 0D49 3005        CMPC5   #VC1NAMLEN,VC1NAM,#^A/ /,-
50 B3 10 0D4E 3006        #16,@CDT$L_RPROCNAME(R3)      ; Is the connection to MSCP$TAPE?
    43 13 0D51 3007        BEQL      CONVCI      ; If yes make connection to VC1
                                0D53 3008
                                0D53 3009        MOVL   (SP),R3      ; Refresh R3 => CDT.
20 C4 AF 03 2D 0D56 3010 TSTVC2:
50 B3 10 0D5B 3011        CMPC5   #VC2NAMLEN,VC2NAM,#^A/ /,-
    27 13 0D5E 3012        #16,@CDT$L_RPROCNAME(R3)      ; Is the connection to DUP?
    53 8ED0 0D60 3013        BEQL      CONVCC      ; If yes make connection to VC2
                                0D60 3014        POPL    R3          ; Remove R3 from stack.
                                0D63 3015        CON_NO_NODE:
00000000'GF 16 0D63 3016        JSB     G^SCS$DEALL_CDT      ; Free R3 => CDT.
50 028C 8F 3C 0D69 3017        MOVZWL  #SS$_NOSUCHNODE,R0      ; Indicate No Such Node.
    05 0D6E 3018        RSB
                                0D6F 3019        CONREJ:
00000000'GF 16 0D6F 3020        JSB     G^SCS$DEALL_CDT      ; Free R3 => CDT.
50 0294 8F 3C 0D75 3021        MOVZWL  #SS$_REJECT,R0      ; Otherwise reject connection
    05 0D7A 3022        RSB
                                0D7B 3023        CON_NO_LISTEN:
00000000'GF 16 0D7B 3024        JSB     G^SCS$DEALL_CDT      ; Free R3 => CDT.
50 215C 8F 3C 0D81 3025        MOVZWL  #SS$_NOLISTENER,R0      ; Indicate No Such Listener.
    05 0D86 3026        RSB
                                0D87 3027
                                0D87 3028        ; Make a connection to MSCP$DISK which is virtual circuit 0 for a
                                0D87 3029        ; UDA.
                                0D87 3030
0D87 3031 CONVC2:
50 00EC C4 9E 0D87 3032        MOVAB   PDT$L_PU_VC2(R4),R0      ; Setup for common code.
```


+ FPC\$CONNECT, COMPLETE PROCESSING A CON

```
51 00F8 C4 9E 0D8C 3033      MOVAB  PDT$W_PU_CRED2(R4),R1  ; Initial credits to assign.
52 02 D0 0D91 3034      MOVL   #2,R2                ; Remote connection ID.
1B 11 0D94 3035      BRB     CON_COMMON          ; Branch around.
      0D96 3036
      0D96 3037 CONVC1:
50 00E8 C4 9E 0D96 3038      MOVAB  PDT$W_PU_VC1(R4),R0      ; Setup for common code.
51 00F6 C4 9E 0D9B 3039      MOVAB  PDT$W_PU_CRED1(R4),R1    ; Initial credits to assign.
52 01 D0 0DA0 3040      MOVL   #1,R2                ; Remote connection ID.
      0C 11 0DA3 3041      BRB     CON_COMMON          ; Branch around.
      0C 11 0DA5 3042
      0DA5 3043 CONVC0:
50 00E4 C4 9E 0DA5 3044      MOVAB  PDT$W_PU_VC0(R4),R0      ; Setup for common code.
51 00F4 C4 9E 0DAA 3045      MOVAB  PDT$W_PU_CRED0(R4),R1    ; Initial credits to assign.
      52 D4 0DAF 3046      CLRL   R2                  ; Remote connection ID is zero.
      0C 11 0DB1 3047 CON_COMMON:
0128 C4 53 8ED0 0DB1 3048      POPL   R3                ; Refresh R3 => CDT.
      52 E1 0DB4 3049      BBC     R2,PDT$B_SERVERS(R4),-    ; See if this Server supported at this
      C1 0DB9 3050      CON_NO_LISTEN                    ; port. If not, branch.
0127 C4 52 E2 0DBA 3051      BBSS   R2,PDT$B_CONBITMAP(R4),-  ; Mark corresponding bit in bit map
      AF 0DBF 3052      CONREJ                      ; as OPEN. If already set, reject.
      60 53 D0 0DC0 3053      MOVL   R3,(R0)              ; Save address of CDT for VCx
      14 A3 52 D0 0DC3 3054      MOVL   R2,CDT$W_RCONID(R3)   ; Set virtual circuit number into CDT
      0126 C4 96 0DC7 3055      INCB   PDT$B_NOCURCON(R4)     ; Increment # of current connections.
      40 A3 61 B0 0DCB 3056
      02 B0 0DCF 3057      MOVW    (R1),CDT$W_SEND(R3)       ; Put in initial send credits.
      28 A3 02 B0 0DCF 3058
      02 B0 0DD1 3059      MOVW    #CDT$C_OPEN,-           ; Move CDT state to
      28 A3 02 B0 0DD1 3059      CDT$W_STATE(R3)           ; open
      0DD3 3060
      0DD3 3061
      0DD3 3062 ; Here we make room on the stack to create a Connect Message to return
      0DD3 3063 ; to the caller. In order to get control after the caller is finished
      0DD3 3064 ; with the Connect data, we call him back as a co-routine.
      0DD3 3065
      51 8ED0 0DD3 3066      POPL   R1                ; R1 has callers return point.
5E D0 AE 9E 0DD6 3067      MOVAB  -SCSCMG$$_SCSCMGDEF(SP),SP    ; Create space on stack.
      52 5E D0 0DDA 3068      MOVL   SP,R2                ; R2=>Connect data area.
      0DD0 3069
      00 6E 3E BB 0DDD 3070      PUSHR  #^M<R1,R2,R3,R4,R5>    ; Save MOVC registers
      62 00 2C 0DDF 3071      MOVCS  #0,(SP),#0,-          ; Zero initialize structure
      30 0DE3 3072      #SCSCMG$$_SCSCMGDEF,(R2)
      3E BA 0DE5 3073      POPR    #^M<R1,R2,R3,R4,R5>    ; Restore MOVC registers
      0DE7 3074
      0DE7 3075 ;
      0DE7 3076 ;
      0DE7 3077      MOVAB  SYSGEN_PARAMETER,-           ; Copy Allocation Class.
      50 01 3C 0DE7 3078      SCSCMG$B_SNDATA+1(R2)
      61 16 0DEA 3079      MOVZWL  S^#SS$_NORMAL,R0        ; Set normal completion
5E 30 AE 9E 0DEC 3080      JSB     (R1)                  ; Call back Caller as co-routine.
      05 0DF0 3081      MOVAB  SCSCMG$$_SCSCMGDEF(SP),SP    ; Free up stack space.
      0DF1 3082      RSB
      0DF1 3083      .DSABL  LSB
      0DF1 3084
```

+ FPC\$DCONNECT, PROCESS A DISCONNECT CALL

.SBTTL + FPC\$DCONNECT, PROCESS A DISCONNECT CALL

ODF1 3086
ODF1 3087
ODF1 3088 :+
ODF1 3089 : FPC\$DCONNECT is called by the SYSAP. It may be called only from the
ODF1 3090 : open state. The CDT is moved to the closed state.
ODF1 3091 :
ODF1 3092 : Inputs:
ODF1 3093 :
ODF1 3094 : R0 -Disconnect reason
ODF1 3095 : R3 -Addr of CDT being disconnected
ODF1 3096 : R4 -Addr of PDT
ODF1 3097 :
ODF1 3098 : Outputs:
ODF1 3099 :
ODF1 3100 : R0 -Status: \$\$\$_NORMAL, \$\$\$_ILLCDTST
ODF1 3101 : R1,R2 -Destroyed
ODF1 3102 : other registers -Preserved
ODF1 3103 : -
ODF1 3104 :
ODF1 3105 :
ODF1 3106 FPC\$DCONNECT::
ODF1 3107

26 A3 50 B0 ODF1 3108 MOVW R0,CDT\$W_REASON(R3) ; Save disconnect reason
ODF5 3109 \$CHK_CDTSTATE = ; Assure that CONNECTION is open.
ODF5 3110 OPEN,-
ODF5 3111 ERROR=STATE_ERR
ODFE 3112

00 B0 ODFE 3113 MOVW #CDT\$C_CLOSED,- ; Move state to closed
28 A3 OE00 3114 CDT\$W_STATE(R3) ;
0126 C4 97 OE02 3115 DECB PDT\$B_NOCURCON(R4) ; Decrement # of current connections.
OE06 3116

50 14 A3 02 00 EF OE06 3117 EXTZV #0,#2,CDT\$L_RCONID(R3),R0; Get index of CDT pointer for connection.
OE0C 3118
00 0127 C4 50 E4 OE0C 3119 BBSC R0,PDT\$B_CONBITMAP(R4),10\$; Clear the connection active bit
OE12 3120 10\$:
OE12 3121

OE12 3122 : The following two instructions save the available credits on the connection
OE12 3123 : that we are disconnecting so that we can later re-connect. This is only
OE12 3124 : useful if dis-connecting and later re-connecting do not have an FPC\$MRESET
OE12 3125 : done in between. FPC\$MRESET re-inits the credits for all possible
OE12 3126 : connections (in INIT_PU_PDT, called from HARDWARE_INIT).
OE12 3127

51 00F4 C440 3E OE12 3128 MOVAW PDT\$W_PU_CRED0(R4)[R0],R1; R1=> Repository of credits for this
OE18 3129 : connection ID.
61 40 A3 B0 OE18 3130 MOVW CDT\$W_SEND(R3),(R1) ; Save current credits for later connect.
OE1C 3131 ; End of code that saves available credits.
OE1C 3132

F2DC CF 9E OE1C 3133 MOVAB NULL_CDT,-
00E4 C440 OE20 3134 PDT\$C_PU_CDTARY(R4)[R0] ; Reset ptr.
00000000'GF 16 OE24 3135 JSB G^SCS\$DEALL_CDT ; Deallocate R3 => CDT.
50 01 3C OE2A 3136 MOVZWL S^#\$\$\$_NORMAL,R0 ; Normal return status.
05 OE2D 3137 RSB ;

SEQUENCED MESSAGE CALLS

```
OE2E 3139      .SBTTL SEQUENCED MESSAGE CALLS
OE2E 3140      .SBTTL + FPC$ALLOCMSG,  ALLOCATE A MESSAGE BUFFER
OE2E 3141
OE2E 3142      ;+
OE2E 3143      ; FPC$ALLOCMSG is optimized for the case where all resources that are
OE2E 3144      ; allocated are available. FPC$ALLOCMSG first checks the state of the
OE2E 3145      ; CONNECTION to assure that it is OPEN. Then it allocates a send credit
OE2E 3146      ; and a buffer. Finally it points R2 at the application (MSCP) portion
OE2E 3147      ; of the buffer, stores R2 in CDRP$L_MSG_BUF and returns a success code.
OE2E 3148
OE2E 3149      ; Exceptions to this flow are handled out of line.
OE2E 3150
OE2E 3151      ; First, if the CONNECTION state is NOT open, we return the SSS_ILLCDTST
OE2E 3152      ; status.
OE2E 3153
OE2E 3154      ; If no send credits are available, the thread is suspended on the
OE2E 3155      ; CDT$L_CRWAITQBL.
OE2E 3156
OE2E 3157      ; Finally, if no buffers are available, we first try to scare some up by
OE2E 3158      ; calling internal subroutine POLL_CMDRING, which polls the command ring
OE2E 3159      ; to free up slots and buffers. If upon return from this call, a buffer
OE2E 3160      ; is available, we simply rejoin the mainline code. If however, buffers
OE2E 3161      ; are still NOT available, we return the previously allocated send credit
OE2E 3162      ; and suspend the thread on PDT$L_BUFQBL.
OE2E 3163
OE2E 3164      ; Upon resumption of threads from either of these wait Q's, we simply
OE2E 3165      ; branch back to the start of the routine and try all over again.
OE2E 3166
OE2E 3167      ; Inputs:
OE2E 3168
OE2E 3169      ; R4                      -Addr of PDT
OE2E 3170      ; R5                      -Addr of CDRP
OE2E 3171      ; CDRP$L_CDT              -Addr of CDT
OE2E 3172
OE2E 3173      ; Outputs:
OE2E 3174
OE2E 3175      ; R0                      -Status: SSS_NORMAL, SSS_ILLCDTST
OE2E 3176      ; R1                      -Destroyed
OE2E 3177      ; R2                      -Addr of message buffer, if status=success
OE2E 3178      ; Other registers         -Preserved
OE2E 3179
OE2E 3180      ; CDRP$L_MSG_BUF          -Addr of message buffer, if status=success
OE2E 3181      ;-
OE2E 3182
OE2E 3183      FPC$ALLOCMSG::
OE2E 3184
OE2E 3185      52    24  A5    D0  OE2E 3186      MOVL    CDRP$L_CDT(R5),R2      ; Get CDT addr
OE2E 3187      OE32 3187      $CHK_CDTSTATE -      ; Verify connection state
OE2E 3188      OE32 3188      OPEN,-      ; is open.
OE2E 3189      OE32 3189      ERROR=STATE_ERR,-      ; Else report error to caller
OE2E 3190      OE32 3190      CDT=R2      ;
OE2E 3191      OE3B 3191
OE2E 3192      OE3B 3192      DECW    CDT$L_SEND(R2)      ; Allocate the send credit.
OE2E 3193      OE3E 3193      BLSS    20$      ; LSS means no credits available.
OE2E 3194      50    0108 D4    0F  OE40 3194      REMQUE  @PDT$L_PU_FQFL(R4),R0      ; R0 => free buffer.
OE2E 3195      OE45 3195      BVS     30$      ; VS implies NO buffers.
```

+ FPC\$ALLOCMSG, ALLOCATE A MESSAGE BUFFE

52	14	A0	9E	0E47	3196	10\$:			
1C	A5	52	D0	0E47	3197		MOVAB	UDABST TEXT(R0),R2	; R2 => MSCP portion of buffer.
	50	01	3C	0E4B	3198		MOVL	R2,CDRPSL MSG_BUF(R5)	; Return to caller in CDRP as well.
			05	0E4F	3199		MOVZWL	S^#SS\$_NORMAL,R0	; Success return.
				0E52	3200		RSB		; And return.
				0E53	3201				
				0E53	3202	20\$:			
51	3C	A2	D0	0E53	3203		MOVL	CDTSL_CRWAITQBL(R2),R1	; R1 => where to INSQUE to await credits.
		13	11	0E57	3204		BRB	40\$; Branch around to common suspend code.
				0E59	3205	30\$:			
	0362		30	0E59	3206		BSBW	POLL_CMDRING	; Reclaim released buffers from
				0E5C	3207				; command ring.
50	0108	D4	0F	0E5C	3208		REMQUE	@PDTSL_PU_FQFL(R4),R0	; Again try for R0 => free buffer.
		E4	1C	0E61	3209		BVC	10\$; VC implies buffers. Branch to mainline.
52	24	A5	D0	0E63	3210		MOVL	CDRPSL CDT(R5),R2	; Refresh R2=>CDT after call to POLL_CMDRING
51	011C	C4	D0	0E67	3211		MOVL	PDTSL_PU_BUFQBL(R4),R1	; R1 => where to INSQUE to await buffers.
				0E6C	3212	40\$:			
	40	A2	B6	0E6C	3213		INCW	CDT\$W_SEND(R2)	; Return improper allocate.
	18	A5	8ED0	0E6F	3214		POPL	CDRPSL_SAVD RTN(R5)	; Save high level return.
				0E73	3215		\$SUSP_SCS	(R1)	; Suspend on R1 => wait Q element.
	18	A5	DD	0E8B	3216		PUSHL	CDRPSL_SAVD RTN(R5)	; Restore high level return.
		9E	11	0E8E	3217		BRB	FPC\$ALLOCMSG	; And go back to check on credits.


```

OE90 3219 .SBTTL + FPC$DEALLOMSG, DEALLOCATE A MESSAGE BUFFER
OE90 3220
OE90 3221 :+
OE90 3222 : FPC$DEALLOMSG resets the message address specified by the caller to
OE90 3223 : the top of the message buffer and clears CDRP$L_MSG_BUF. It then
OE90 3224 : decides whether to insert the free buffer onto the RESPONSE RING
OE90 3225 : or onto the free Q of buffers, with the RESPONSE RING having priority,
OE90 3226 : and being selected whenever it is not entirely full. If the RESPONSE
OE90 3227 : RING is full then the buffer is INSQUEd onto PDT$L_PU_FQBL.
OE90 3228 :
OE90 3229 : If this INSQUE represents the first buffer on the free Q, then we
OE90 3230 : attempt to resume any threads waiting for buffers (PDT$L_PU_BUFQFL).
OE90 3231 : While the free Q remains non-empty and there exist threads waiting
OE90 3232 : for buffers, we continue to resume the threads until either we run
OE90 3233 : out of buffers or we run out of threads to resume.
OE90 3234 :
OE90 3235 : Internal entrypoint Q_DEALLOC_BUF is called from POLL_CMDRING after
OE90 3236 : R2 has been pointed at the buffer header. Also internal entrypoint
OE90 3237 : INSERT_IN_RRING is called from INIT_PU_PDT at CONNECT time in order to
OE90 3238 : prime the RESPONSE RING.
OE90 3239 : Inputs:
OE90 3240 :
OE90 3241 : R4 -Addr of PDT
OE90 3242 : R5 -Addr of CDRP
OE90 3243 : CDRP$L_MSG_BUF -Addr of message buffer
OE90 3244 :
OE90 3245 : Outputs:
OE90 3246 :
OE90 3247 : R0-R2 -Destroyed
OE90 3248 : Other registers -Preserved
OE90 3249 : CDRP$L_MSG_BUF -Cleared
OE90 3250 :-
OE90 3251 :
OE90 3252 FPC$DEALRGMSG::
52 14 C2 OE90 3253 SUBL #UDAB$T TEXT,R2 ; R2 => buffer header.
08 11 OE93 3254 BRB Q_DEALLOC_BUF ; and branch around to common code.
OE95 3255 FPC$DEALLOMSG::
OE95 3256
52 14 C3 OE95 3257 SUBL3 #UDAB$T TEXT,-
1C A5 OE97 3258 CDRP$L_MSG_BUF(R5),R2 ; R2 => buffer header.
1C A5 D4 OE9A 3259 CLRL CDRP$L_MSG_BUF(R5) ; Prevent spurious deallocates.
OE9D 3260
OE9D 3261 FPC$QUEUEDG::
OE9D 3262 Q_DEALLOC_BUF: ; Called here from POLL_CMDRING,
OE9D 3263 ; with R2 => free buffer.
0125 10 91 OE9D 3264 CMPB #UDASK_RINGSIZE,-
C4 OE9F 3265 PDT$B_RRINGCNT(R4) ; See if response ring is full up.
2A 15 OEA2 3266 BLEQ INSERT_IN_FREEQ ; LEQ implies full. Goto put on free Q.
OEA4 3267
OEA4 3268 INSERT_IN_RRING: ; Here R2 => buffer to insert.
OEA4 3269
50 04 00 EF OEA4 3270 EXTZV #0,#UDASK_RINGEXP,-
0123 C4 OEA7 3271 PDT$B_RRINGINX(R4),R0 ; R0 = index of slot to use in response
; ring.
OEA8 3272
0A A2 90 OEA8 3273 MOVB UDAB$B_BUFFNO(R2),-
013C C440 OEA8 3274 PDT$B_RRCONTENT(R4)[R0] ; Label new contents of this slot.
OEB2 3275

```

+ FPC\$DEALLMSG, DEALLOCATE A MESSAGE BU

08 A2	50	90	0EB2	3276	MOVB	R0,UDAB\$B_RINGINX(R2)	; Remember where this buffer is for
09 A2	01	90	0EB6	3277	MOVB	#1,UDAB\$B_RINGNO(R2)	; debugging.
			0EBA	3278			
			0EBA	3279	ASSUME	UDAB\$B_CREDTYPE EQ	UDAB\$W_MSG_LEN+2
			0EBA	3280	ASSUME	UDAB\$B_CONID EQ	UDAB\$B_CREDTYPE+1
	3C	D0	0EBA	3281	MOVL	#UDAB\$C_LENGTH-UDAB\$T_TEXT,-	; Initialize response buffer
10 A2			0EBC	3282		UDAB\$W_MSG_LEN(R2)	; length and zero credits,
			0EBE	3283			; type and conid.
			0EBE	3284			
0C A2		D0	0EBE	3285	MOVL	UDAB\$L_DESCRIP(R2),-	; Fill in command ring slot.
0208 C440			0EC1	3286		PDT\$L_RSPRING(R4)[R0]	
0123 C4		96	0EC5	3287	INCB	PDT\$B_RRINGINX(R4)	; Next time use next slot in ring.
0125 C4		96	0EC9	3288	INCB	PDT\$B_RRINGCNT(R4)	; Increment # slots in use.
			0ECD	3289			
			0ECD	3290	INSERT_RTN:		
		05	0ECD	3291	RSB		; And return to caller or caller's caller.
			0ECE	3292			
			0ECE	3293	INSERT_IN_FREEQ:		
			0ECE	3294			
	62	0E	0ECE	3295	INSQUE	UDAB\$L_FLINK(R2),-	; Insert free buffer onto free Q.
010C D4			0ED0	3296		@PDT\$L_PU_FQBL(R4)	
F8		12	0ED3	3297	BNEQ	INSERT_RTN	; NEQ implies not first buffer on Q.
			0ED5	3298			
			0ED5	3299	10\$: \$RESUME_FP	-	; Resume first thread awaiting buffers.
			0ED5	3300		@PDT\$L_PU_BUFQFL(R4),-	
			0ED5	3301		QEMPTY=INSERT_RTN	; Label to branch to if Q empty.
			0EE9	3302			
0104 C4		D1	0EE9	3303	CMPL	PDT\$L_PU_FQPTR(R4),-	; Test for emptiness of free Q.
0108 C4			0EED	3304		PDT\$L_PU_FQFL(R4)	; If list points to itself, then empty.
E3		12	0EF0	3305	BNEQ	10\$; NEQ means NOT empty, so try to resume
		05	0EF2	3306	RSB		; Return to caller if empty.

+ FPC\$RCHMSGBUF, RECYCLE MESSAGE BUFFER

```

OEF3 3308      .SBTTL +      FPC$RCHMSGBUF, RECYCLE MESSAGE BUFFER
OEF3 3309      .SBTTL +      AT HIGH PRIORITY
OEF3 3310      .SBTTL +      FPC$RCLMSGBUF, RECYCLE MESSAGE BUFFER
OEF3 3311      .SBTTL +      AT LOW PRIORITY
OEF3 3312
OEF3 3313
OEF3 3314      :+ FPC$RCxMSGBUF checks if there is at least one send credit. If
OEF3 3315      :not, the SYSAP is suspended until there is. FPC$RCxMSGBUF then
OEF3 3316      :decrements the send credit. The wait, if required, places the
OEF3 3317      :SYSAP CDRP at the end of the wait queue for low priority and at
OEF3 3318      :the head of the queue for high priority. The address of the
OEF3 3319      :buffer being recycled is returned in R2.
OEF3 3320
OEF3 3321      Inputs:
OEF3 3322
OEF3 3323      R4                      -Addr of PDT
OEF3 3324      R5                      -Addr of CDRP
OEF3 3325      CDRP$L_MSG_BUF        -Addr of message buffer
OEF3 3326      CDRP$L_CDT          -Addr of CDT
OEF3 3327
OEF3 3328      Outputs:
OEF3 3329
OEF3 3330      R0                      -Status: SS$_NORMAL, SS$_ILLCDTST
OEF3 3331      R1                      -Destroyed
OEF3 3332      R2                      -Addr of message buffer
OEF3 3333      Other registers        -Preserved
OEF3 3334      :-
OEF3 3335
OEF3 3336
OEF3 3337 FPC$RCHMSGBUF::
OEF3 3338
51 24 A5 D0 OEF3 3339      MOVL      CDRP$L_CDT(R5),R1      ; R1 => CDT.
OEF7 3340      $CHK_CDTSTATE -      ; Assure that connection open.
OEF7 3341      OPEN,-
OEF7 3342      ERROR=STATE_ERR,-
OEF7 3343      CDT=R1
OEF7 3344      40 A1 B7 OF00 3344      DECW      CDT$W_SEND(R1)      ; Test decrement a credit.
OEF7 3345      08 19 OF03 3345      BLSS      10$      ; LSS implies there were none to give.
52 1C A5 D0 OF05 3346      MOVL      CDRP$L_MSG_BUF(R5),R2      ; Return R2 => msg buffer.
50 01 3C OF09 3347      MOVZWL   S^#SS$_NORMAL,R0      ; Return status for caller
OEF7 3348      05 OF0C 3348      RSB      ; And return.
OEF7 3349      OF0D 3349      10$:      ; Here only if allocation failure.
OEF7 3350      40 A1 B6 OF0D 3350      INCW      CDT$W_SEND(R1)      ; Restore from above test decrement.
OEF7 3351      18 A5 8ED0 OF10 3351      POPL     CDRP$L_SAVD_RTN(R5)      ; Save first level return address.
OEF7 3352      OF14 3352      $SUSP_SCS -      ; Suspend this thread at HEAD of
OEF7 3353      OF14 3353      CDT$L_CRWAITQFL(R1)      ; allocation wait list.
OEF7 3354      18 A5 DD OF2D 3354      PUSHL    CDRP$L_SAVD_RTN(R5)      ; Restore first level return to stack.
OEF7 3355      C1 11 OF30 3355      BRB      FPC$RCHMSGBUF      ; Go back and try to allocate.
OEF7 3356      OF32 3356
OEF7 3357 FPC$RCLMSGBUF::
OEF7 3358
51 24 A5 D0 OF32 3359      MOVL      CDRP$L_CDT(R5),R1      ; R1 => CDT.
OEF36 3360      $CHK_CDTSTATE -      ; Assure that connection open.
OEF36 3361      OPEN,-
OEF36 3362      ERROR=STATE_ERR,-
OEF36 3363      CDT=R1
OEF36 3364      40 A1 B7 OF3F 3364      DECW      CDT$W_SEND(R1)      ; Test decrement a credit.
```

+ AT LOW PRIORITY

```

52 1C 08 19 0F42 3365
50 01 00 0F44 3366
    01 03 0F48 3367
    05 0F4B 3368
    0F4C 3369 10$:
40 A1 B6 0F4C 3370
18 A5 8ED0 0F4F 3371
    0F53 3372
    0F53 3373
18 A5 DD 0F6C 3374
C1 11 0F6F 3375

```

```

BLSS 10$
MOVL CDRP$L_MSG_BUF(R5),R2
MOVZWL S^#SS$ _NORMAL,R0
RSB

INCW CDT$W_SEND(R1)
POPL CDRP$L_SAVD_RTN(R5)
SSUSP_SCS -
@CDT$L_CRWAITQBL(R1)
PUSHL CDRP$L_SAVD_RTN(R5)
BRB FPC$RCLMSGBUF

```

```

; LSS implies there were none to give.
; Return R2 => msg buffer.
; Return status for caller
; And return.
; Here only if allocation failure.
; Restore from above test decrement.
; Save first level return address.
; Suspend this thread at TAIL of
; allocation wait list.
; Restore first level return to stack.
; Go back and try to allocate.

```


+ FPC\$SND CNTMSG, SEND COUNTED SEQUENCED

.SBTTL + FPC\$SND CNTMSG, SEND COUNTED SEQUENCED MESSAGE

OF71	3377	:	Inputs:		
OF71	3378	:			
OF71	3379	:			
OF71	3380	:	R1	-# bytes of application data	
OF71	3381	:	R4	-Addr of PDT	
OF71	3382	:	R5	-Addr of CDRP	
OF71	3383	:	CDRPSL_CDT(R5)	-Addr of CDT	
OF71	3384	:	CDRPSL_MSG_BUF(R5)	-Addr of message (user portion)	
OF71	3385	:	CDRPSL_RSPID(R5)	-RSPID (to set RETFLG)	
OF71	3386	:			
OF71	3387	:	Outputs:		
OF71	3388	:			
OF71	3389	:	R0	-Status: \$\$\$_NORMAL, \$\$\$_ILLCDTST	
OF71	3390	:	R1,R2	-Destroyed	
OF71	3391	:	Other registers	-Preserved	
OF71	3392	:			
OF71	3393	:			
OF71	3394	:	FPC\$SND CNTMSG::		
OF71	3395	:			
50	24 A5 D0	OF71	3396	MOVL CDRPSL_CDT(R5),R0	: R0 => CDT
		OF75	3397	\$CHK_CDT STATE -	: Verify connection is
		OF75	3398	OPEN,-	: open
		OF75	3399	ERROR=STATE_ERR,-	: Else report error to SYSAP
		OF75	3400	CDT=R0	
		OF7E	3401		
52	14 C3	OF7E	3402	SUBL3 #UDAB\$T TEXT,-	: Point to buffer header.
1C A5		OF80	3403	CDRPSL_MSG_BUF(R5),R2	: R2 => buffer header.
1C A5 D4		OF83	3404	CLRL CDRPSL_MSG_BUF(R5)	: Prevent spurious deallocates.
		OF86	3405		
		OF86	3406	ASSUME UDAB\$B_CREDTYPE	EQ UDAB\$W_MSG_LEN+2
		OF86	3407	ASSUME UDAB\$B_CONID	EQ UDAB\$B_CREDTYPE+1
10 A2 51 D0		OF86	3408	MOVL R1,UDAB\$W_MSG_LEN(R2)	: Put message length in header.
14 A0 90		OF8A	3409	MOVB CDT\$R_CONID(R0),-	: Put remote connection ID in message
13 A2		OF8D	3410	UDAB\$B_CONID(R2)	: header.
		OF8F	3411		
10 A5 53 7D		OF8F	3412	MOVQ R3,CDRPSL_FR3(R5)	
OC A5 8ED0		OF93	3413	POPL CDRPSL_FPC(R5)	: Save context in CDRP.
		OF97	3414		
		OF97	3415	CMPB #UDASK_RINGSIZE,-	: See if any slots available in command
0122 C4 91		OF99	3416	PDT\$B_CRINGCNT(R4)	: ring.
2A 15		OF9C	3417	BLEQ CRING_FULL	: LEQ implies ring full.
		OF9E	3418		
		OF9E	3419	INSERT_IN_CRING:	: Called from POLL_CMDRING with
		OF9E	3420		: R2 => buffer.
		OF9E	3421		
50 04 00 EF		OF9E	3422	EXTZV #0,#UDASK_RINGEXP,-	: R0 = index of slot to use in command
0120 C4		OFA1	3423	PDT\$B_CRINGINX(R4),R0	: ring.
		OFA5	3424		
		OFA5	3425		
		OFA5	3432		
OA A2 90		OFA5	3433	MOVB UDAB\$B_BUFFNO(R2),-	: Label new contents of this slot.
012C C440		OFA8	3434	PDT\$B_CRCONTENT(R4)[R0]	
		OFA8	3435		
		OFA8	3436	ASSUME UDAB\$B_RINGNO EQ UDAB\$B_RINGINX+1	
08 A2 50 9B		OFA8	3437	MOVZBW R0,UDAB\$B_RINGINX(R2)	: Remember where this buffer is for
		OFA8	3438		: debugging.
		OFA8	3439		


```

      OC A2  D0  OFB0 3440 ENABLE_COMMAND_START:
0248 C440      OFB0 3441      MOVL  UDAB$ DESCRIP(R2),-      ; Fill in command ring slot.
      OFB3 3442      PDT$ CMDRING(R4)[R0]      ; NOTE, this instruction copied below
      OFB7 3443      ; to allow for dynamic patching to
      OFB7 3444      ; enable tracing.
51  0100 C4  D0  OFB7 3445 ENABLE_COMMAND_END:
      50  61  B0  OFB7 3446      MOVL  PDT$ PU_CSR(R4),R1      ; R1 => UDA CSR.
      OF9C 3447      MOVW   UDAIP(R1),R0      ; Jiggle controller register to force
      OFBF 3448      ; polling of command ring.
      0120 C4  96  OFBF 3449      INCB   PDT$B_CRINGINX(R4)      ; Next time use next slot in ring.
      0122 C4  96  OFC3 3450      INCB   PDT$B_CRINGCNT(R4)      ; Increment # slots in use.
      OFC7 3451
      05  OFC7 3452      RSB      ; And return to caller or caller's caller.
      OFC8 3453
      OFC8 3454 CRING_FULL:
      OFC8 3455
      OFC8 3456 ; The following instructions (commented out) were useful in debugging only.
      OFC8 3457 ; BEQL 10$      ; EQL means ring full not overflowed.
      OFC8 3458 ; BUG_CHECK UDAPORT,FATAL      ; Here we have overflowed command ring.
      OFC8 3459 ; 10$:
      62  0E  OFC8 3460      INSQUE UDAB$ FLINK(R2),-      ; Insert onto tail of backed up buffers.
0114 D4      OFCA 3461      @PDT$ PU_SNDQBL(R4)
      01EE 30  OFCD 3462      BSBW   POLL_CMDRING      ; Free up any possible slots in
      OFD0 3463      ; command ring and dequeue from SND Q.
      05  OFD0 3464      RSB      ; Return to caller's caller.
      OFD1 3465
      OFD1 3466 ENABLE_COMMAND_CODE:      ; If we enable tracing, the driver will
      OFD1 3467      ; dynamically patch location
      OFD1 3468      ; ENABLE_COMMAND_START to BRW here.
      F4A2 30  OFD1 3469      BSBW   TRACE_COMMAND      ; Copy command buffer to trace table.
      OFD4 3470
      OC A2  D0  OFD4 3471      MOVL  UDAB$ DESCRIP(R2),-      ; Fill in command ring slot.
0248 C440      OFD7 3472      PDT$ CMDRING(R4)[R0]
      FFD9 31  OFDB 3473      BRW    ENABLE_COMMAND_END      ; Branch back to normal stream.
      OFDE 3474
      0000001E OFDE 3475 ENABLE_COMMAND_OFFSET=ENABLE_COMMAND_CODE-ENABLE_COMMAND_START-3
```


+ FPC\$MAPIRP, Map a buffer

```
OFDE 3477 .SBTTL + FPC$MAPIRP, Map a buffer
OFDE 3478
OFDE 3479 :+
OFDE 3480 : FPC$MAPIRP - map a user buffer given IRP (CDRP) values.
OFDE 3481 :
OFDE 3482 : Inputs:
OFDE 3483 : R4 => PDT
OFDE 3484 : R5 => CDRP
OFDE 3485 : CDRP$L_LBUFH_AD => area to fill in with UNIBUS virtual address of
OFDE 3486 : user buffer.
OFDE 3487 :
OFDE 3488 : Outputs:
OFDE 3489 : CDRP$L_UBARSRCE filled in with a) datapath assigned, b) # UBA map
OFDE 3490 : registers assigned, c) 1st map register
OFDE 3491 :
OFDE 3492 : @CDRP$L_LBUFH_AD filled in with UNIBUS virtual address of user buffer
OFDE 3493 :
OFDE 3494 : R0-R2 destroyed
OFDE 3495 : Other registers preserved.
OFDE 3496 :-
OFDE 3497
OFDE 3498 FPC$MAPIRP::
OFDE 3499 CLRL CDRP$L_UBARSRCE(R5) ; Initialize.
OFDE 3500 POPL CDRP$L_SAVD_RTN(R5) ; Save return for two level process.
OFDE 3501
OFDE 3502 JSB @REQDATAPATH_TV ; Allocate datapath, if any available.
OFDE 3503
OFDE 3504 JSB G^IOC$REQMAPUDA ; Allocate map registers.
OFDE 3505 JSB G^IOC$LUBAUDAMAP ; Load map registers for this transfer.
OFDE 3506
OFDE 3507 : Here we fill in the local buffer handle. A description of the UBA mapping
OFDE 3508 : resources assigned to this transfer are currently in CDRP$L_UBARSRCE.
OFDE 3509 : We calculate the UNIBUS virtual address of the transfer and put this
OFDE 3510 : value into the @CDRP$L_LBUFH_AD.
OFDE 3511 :
OFDE 3512 MOVZWL CDRP$L_BOFF(R5),R0 ; Calculate UNIBUS virtual address of
OFDE 3513 : user buffer. First get byte offset.
OFDE 3514
OFDE 3515 INSV CDRP$L_UBARSRCE+UBMD$B_DATAPATH(R5),-
OFDE 3516 : #24,#8,R0 ; Place datapath number in high byte.
OFDE 3517 BICB #1,R0 ; Clear low bit in case of odd address.
OFDE 3518
OFDE 3519 INSV CDRP$L_UBARSRCE+UBMD$W_MAPREG(R5),-
OFDE 3520 : #9,#9,R0 ; High order of UNIBUS virtual
OFDE 3521 : address is map register #.
OFDE 3522 MOVL CDRP$L_LBUFH_AD(R5),R1 ; R1 => destination for buffer handle.
OFDE 3523
OFDE 3524 MOVL R0,(R1)+ ; Write 'UBA' buffer
OFDE 3525 CLRQ (R1) ; handle and zero out rest.
OFDE 3526
OFDE 3527 JMP @CDRP$L_SAVD_RTN(R5) ; Return to top level caller.
```

3C A5 D4
18 A5 8ED0

F1B3 DF 16
00000000'GF 16
00000000'GF 16

50 D0 A5 3C
50 08 18 F0
50 01 8A
50 3C A5 F0
50 09 09
51 2C A5 D0
81 50 D0
61 7C
18 B5 17

+ FPC\$MAPIRP_UV1, Map a buffer for uVAX

.SBTTL + FPC\$MAPIRP_UV1, Map a buffer for uVAX I

1014 3529
1014 3530
1014 3531
1014 3532
1014 3533
1014 3534
1014 3535
1014 3536
1014 3537
1014 3538
1014 3539
1014 3540
1014 3541
1014 3542
1014 3543
1014 3544
1014 3545
1014 3546
1014 3547
1014 3548
1014 3549
1014 3550
1014 3551
1017 3552
101A 3553
101B 3554
101B 3555
101B 3556
101F 3557
101F 3558
1025 3559
102B 3560
102B 3561
102B 3562
102B 3563
102B 3564
102B 3565
102B 3566
102F 3567
102F 3568
102F 3569
1032 3570
1035 3571
1035 3572
1039 3573
1039 3574
1041 3575
1046 3576
1048 3577
1048 3578

FPC\$MAPIRP_UV1 - map a user buffer given IRP (CDRP) values.

Inputs:

R4 => PDT

R5 => CDRP

CDRP\$L_LBUFH_AD => area to fill in with UNIBUS virtual address of
user buffer.

Outputs:

CDRP\$L_UBARSRC filled in with a) datapath assigned, b) # UBA map
registers assigned, c) 1st map register

@CDRP\$L_LBUFH_AD filled in with UNIBUS virtual address of user buffer

R0-R2 destroyed

Other registers preserved.

FPC\$MAPIRP_UV1::

CLRL CDRP\$L_UBARSRC(R5)

; Initialize.

BLBS CDRP\$W_BOFF(R5),-

; Branch around if Not word aligned.

MAP_UNALIGN

; Label to allow branch back.

MAP_ODD:

POPL CDRP\$L_SAVD_RTN(R5)

; Save return for two level process.

JSB G^IOC\$REQMAPUDA

; Allocate map registers.

JSB G^IOC\$LUBAUDAMAP

; Load map registers for this transfer.

: Here we fill in the local buffer handle. A description of the UBA mapping
resources assigned to this transfer are currently in CDRP\$L_UBARSRC.
We calculate the UNIBUS virtual address of the transfer and put this
value into the @CDRP\$L_LBUFH_AD.

MOVZWL CDRP\$W_BOFF(R5),R0

; Calculate UNIBUS virtual address of
; user buffer. First get byte offset.

INSV CDRP\$L_UBARSRC+UBMD\$W_MAPREG(R5),-

; High order of UNIBUS virtual
; address is map register #.

MOVL CDRP\$L_LBUFH_AD(R5),R1

; R1 => destination for buffer handle.

BISL3

#UQPORT M MAPPED,R0,(R1)+; Write BUS virtual address of buffer

MOVL PUSL_MPHYAD,(R1)+

; and physical address of the pseudo

CLRL

(R1); map registers and zero out rest.

JMP @CDRP\$L_SAVD_RTN(R5)

; Return to top level caller.

3C A5 D4
D0 A5 E8
30

18 A5 8ED0

00000000'GF 16
00000000'GF 16

50 D0 A5 3C

50 09 09 3C A5 F0

51 2C A5 D0

81 50 80000000 8F C9
81 F08F CF D0

61 D4

18 B5 17

+ MAP_UNALIGN uVAX I Q-BUS Map Unaligned

```
104B 3580 .SBTTL + MAP_UNALIGN uVAX I Q-BUS Map Unaligned Buffer
104B 3581 :+
104B 3582 : MAP_UNALIGN - routine to allocate the Aligned Page (i.e. 512 physically
104B 3583 : contiguous bytes) and to copy upto 512 bytes of user data to this
104B 3584 : buffer. Things to keep in mind:
104B 3585 :
104B 3586 : 1. Allocation of the Aligned Page is governed by PDT$PQ_POWNER. This
104B 3587 : longword, if zero, indicates the Aligned Page is available. Non-zero
104B 3588 : implies it is allocated and the owner's CDRP is saved in the longword.
104B 3589 :
104B 3590 : 2. PDT$PQ_PGQFL, PDT$PQ_PGQBL are a queue header that lists CDRP's
104B 3591 : waiting for the Aligned Page.
104B 3592 :
104B 3593 : 3. PDT$PQ_SVPSTE points to a PTE slot in the System Page table that
104B 3594 : belongs to this Port. PDT$PQ_UBFSVA contains the System Virtual
104B 3595 : Address associated with this PTE.
104B 3596 :
104B 3597 : 'Mapping' an unaligned user buffer means copying all, or the first 512 bytes,
104B 3598 : whichever is less, of the user data to the Aligned Page. This means that
104B 3599 : CDRP$BCNT (if originally >512) is reduced to 512. The copying uses
104B 3600 : the PTE slot pointed at by PDT$PQ_SVPSTE, and the System Virtual Address
104B 3601 : associated with it. Essentially, the PTE that points to the first segment
104B 3602 : (i.e. the user data in the first page of the user's buffer) is put into
104B 3603 : our reserved PTE slot. Then the user data in this segment is copied. Then
104B 3604 : the next user PTE is loaded into our reserved PTE slot and the second (if
104B 3605 : any) segment is copied.
104B 3606 :
104B 3607 : Inputs:
104B 3608 : R4 => PDT
104B 3609 : R5 => CDRP
104B 3610 :
104B 3611 : Outputs:
104B 3612 : Aligned Page allocated and user data copied.
104B 3613 : Registers R0-R2 modified.
104B 3614 :-
104B 3615 :
104B 3616 MAP_UNALIGN:
104B 3617 MOVL PDT$UCB0(R4),R0 ; We come here for unaligned transfers.
1050 3618 BBS #PS1 0,OD,- ; R0 => port UCB.
1052 3619 UCB$Q_PORTSTEP1(R0),- ; If controller supports odd addresses,
1055 3620 MAP_ODD ; then branch back in line.
1056 3621 TSTL PDT$PQ_POWNER(R4) ; Test if aligned page available.
105A 3622 BEQL 10$ ; EQL implies available.
105C 3623 MOVQ R3,CDRP$R3(R5) ; Else save context.
1060 3624 INCW @CDRP$R0CPT(R5) ; Bump UCB$R_WAITCNT.
1063 3625 POPL CDRP$R_FPC(R5) ; Save return point.
1067 3626 INSQUE CDRP$R_FQFL(R5),- ; Queue this CDRP to resource wait Q.
1069 3627 @PDT$PQ_PGQBL(R4)
106C 3628 RSB ; Return to caller's caller.
106D 3629 10$:
106D 3630 MOVL R5,PDT$PQ_POWNER(R4) ; Allocate Aligned Page.
1072 3631 CMPL #NO_PHYCONTIGBYT,- ; Test for transfer shorter than length
1078 3632 CDRP$BCNT(R5) ; of aligned contiguous buffer.
107A 3633 BGEQ 20$ ; GEQ implies short transfer.
107C 3634 MOVL #NO_PHYCONTIGBYT,- ; Reduce transfer to total bytes in
1082 3635 CDRP$BCNT(R5) ; aligned contiguous buffer.
1084 3636 20$:
50 00DC C4 D0 104B 3617
07 E0 1050 3618
00AC C0 1052 3619
C5 1055 3620
OCA0 C4 D5 1056 3621
11 13 105A 3622
10 A5 53 7D 105C 3623
28 B5 B6 1060 3624
0C A5 8ED0 1063 3625
65 0E 1067 3626
OC9C D4 1069 3627
05 106C 3628
106D 3629
OCA0 C4 55 D0 106D 3630
00000200 8F D1 1072 3631
D2 A5 1078 3632
08 18 107A 3633
00000200 8F D0 107C 3634
D2 A5 1082 3635
1084 3636
```

+ MAP_UNALIGN uVAX I Q-BUS Map Unaligned

```

1084 3637
1084 3638 ; Here after successfully allocating the aligned buffer, if the user operation
1084 3639 ; is a WRITE, we copy the user's data to the aligned buffer. If the user
1084 3640 ; operation is a READ, we branch around the copy and merely fillin the
1084 3641 ; CDRP$L_LBUFH_AD field. The data from a READ is copied to the user
1084 3642 ; buffer after the I/O is complete.
1084 3643
1084 3644 BBS #IRP$V_FUNC,= ; If a READ, branch around.
11 CA A5 E0 1086 3645 CDRP$W_STS(R5),40$
1B 10 1089 3646 BSBB SETUP_COPY_SEG1 ; Call to setup copy of user data.
108B 3647 ; This is a complex routine that
108B 3648 ; returns: R0 = # bytes in first
108B 3649 ; segment, R1 => user data,
108B 3650 ; R2 => aligned page, (SP) = # bytes
108B 3651 ; in second segment (or zero if none),
108B 3652 ; 4(SP) (valid only if (SP) non-zero)
108B 3653 ; zero.
82 81 90 108B 3654 30$:
FA 50 F5 108E 3655 MOVB (R1)+,(R2)+ ; Copy a byte.
50 8ED0 1091 3656 SOBGTR R0,30$ ; Loop thru entire segment.
04 13 1094 3657 POPL R0 ; Get length of next segment (if any).
77 10 1096 3658 BEQL 40$ ; EQL implies no more.
F1 11 1098 3659 BSBB SETUP_COPY_SEG2 ; Call to setup copy of second segment.
109A 3660 BRB 30$ ; Branch back to loop.
51 2C A5 D0 109A 3661 40$:
81 OCA4 C4 D0 109E 3662 MOVL CDRP$L_LBUFH_AD(R5),R1 ; R1 => destination for buffer handle.
61 05 10A3 3663 MOVL PDT$L_PQ_PGPHAD(R4),(R1)+ ; Copy physical address of Aligned Page
10A5 3664 CLRQ (R1) ; and zero out rest.
10A5 3665 RSB ; And return to caller.

```


SETUP_COPY_SEG1 and SETUP_COPY_SEG2

10A6 3667
10A6 3668
10A6 3669
10A6 3670
10A6 3671
10A6 3672
10A6 3673
10A6 3674
10A6 3675
10A6 3676
10A6 3677
10A6 3678
10A6 3679
10A6 3680
10A6 3681
10A6 3682
10A6 3683
10A6 3684
10A6 3685
10A6 3686
10A6 3687
10A6 3688
10A6 3689
10A6 3690
10A6 3691
10A6 3692
10A6 3693
10A6 3694
10A6 3695
10A6 3696
10A6 3697
10A6 3698
10A6 3699
10A6 3700
10AA 3701
10AA 3702
10AD 3703
10B0 3704
10B3 3705
10B5 3706
10B8 3707
10B9 3708
10B9 3709
10B9 3710
10C1 3711
10C5 3712
10CB 3713
10CE 3714
10D0 3715
10D5 3716
10DC 3717
10DE 3718
10E6 3719
10EE 3720
10EE 3721
10F0 3722
10F2 3723

.SBTTL SETUP_COPY_SEG1 and SETUP_COPY_SEG2

SETUP_COPY_SEG1 - a routine to setup a copy from (to) the aligned
page to (from) the user's buffer.

Inputs:

CDRPSL_SVAPTE => System Virtual Address of the Page Table Entry that
maps the first page of the user buffer into user
space.
CDRPSW_BOFF => Offset in this page of user data.
PDTSL_PQ_SVPSTE => System Virtual Address of an available PTE in the
System Page Table that we can use to map the user
buffer into System space.
PDTSL_PQ_UBFSVA => System space address of the page associated with
previous PTE.
PDTSL_PQ_ALGNPG => System space address of the aligned page.
CDRPSL_BCNT => Length of data to transfer. (< or = 512)

Outputs:

R0 = Number of bytes to copy in first segment. The first segment is
defined by the data beginning at CDRPSW_BOFF in the first
page of the user buffer and continuing till the end of the page.
R1 => System space address of first byte in user buffer.
R2 => First byte of aligned page.
(SP) = Either # bytes in the second segment or zero.
4(SP) = Valid only if (SP) is non-zero. Then this is zero.

Note: CDRPSL_SAVD_RTN is modified.

SETUP_COPY_SEG1:

18 A5 8ED0	10A6 3700	POPL	CDRPSL_SAVD_RTN(R5)	: Clean stack so that we can return
	10AA 3701			: values to caller on it.
CC A5 D0	10AA 3702	MOVL	CDRPSL_SVAPTE(R5),-	: Save user SVAPTE in PDT.
OCB0 C4	10AD 3703		PDTSL_PQ_USRSTE(R4)	
15 00 EF	10B0 3704	EXTZV	#PTE\$V PFN,#PTE\$S PFN,-	: Construct a PTE with the proper PFN,
CC B5	10B3 3705		@CDRPSL_SVAPTE(R5),-	: protection=KW, and own=K. Here we
OCAC D4	10B5 3706		@PDTSL_PQ_SVPSTE(R4)	: move the PFN.
	10B8 3707	BISL	#PTE\$M_VALID!-	: And here we set VALID, the protection
	10B9 3708		PTE\$C_KW!-	: and the ownership and thereby map
	10B9 3709		PTE\$C_KOWN,-	: the first page of the user buffer
OCAC D4 90000000 8F	10B9 3710		@PDTSL_PQ_SVPSTE(R4)	: into System Space.
51 50 OCA8 C4	10C1 3711	MOVZWL	CDRPSW_BOFF(R5),R0	: R0 = BOFF.
	10C5 3712	ADDL3	PDTSL_PQ_UBFSVA(R4),R0,R1	: R1 = SVA of 1st byte of user data.
	10CB 3713	INVALID	R1	: Invalidate virtual address.
	10CE 3714	CLRL	-(SP)	: Place signal on top of stack.
52 D2 A5 50	10D0 3715	ADDL3	R0,CDRPSL_BCNT(R5),R2	: R2 = relative offset of end of buffer
52 00000200 8F	10D5 3716	CMPL	#512,R2	: Does user buffer slop onto next pages
	10DC 3717	BGEQ	20\$: GEQ implies no spill over.
50 00000200 8F	10DE 3718	SUBL3	R0,#512,R0	: R0 = length of 1st segment to copy.
7E 52 FFFFFFFE00 8F	10E6 3719	BICL3	#^C<^X1FF>,R2,-(SP)	: Push length of spill into last page
	10EE 3720			: of user buffer.
	10EE 3721	BNEQ	5\$: NEQ implies there was such slop.
	10F0 3722	TSTL	(SP)+	: If no such slop, clear top of stack.
	10F2 3723		5\$:	

SETUP_COPY_SEG1 and SETUP_COPY_SEG2

```

52 52 F7 8F 78 10F2 3724      ASHL    #-9,R2,R2      ; R2 now contains one more than the
                                     ; number of full (512 byte) pages
                                     ; in middle of user buffer that have
                                     ; not yet been accounted for. Note
                                     ; that the first page (possibly
                                     ; partially filled - CDRP$W_BOFF)
                                     ; is accounted for in R0, and any
                                     ; spill over into the last page is
                                     ; accounted for on the top of stack.
                                     10$:
                                     52 D7 10F7 3734      DECL    R2      ; Reduce # full pages by one.
                                     OC 15 10F9 3735      BLEQ    30$      ; If zero, branch around.
00000200 8F DD 10FB 3736      PUSHL   #512      ; Indicate full 512 byte segment.
                                     F4 11 1101 3737      BRB     10$      ; Branch back.
                                     1103 3738      20$:
50 D2 A5 D0 1103 3739      MOVL     CDRP$L_BCNT(R5),R0      ; R0 = length of 1st and only segment.
                                     1107 3740      30$:
                                     1107 3741
                                     1107 3742      ; Here R0 has the length of the first segment to copy. The lengths of the
                                     1107 3743      ; subsequent segments have all been pushed onto the stack in inverse order.
                                     1107 3744      ; The last item pushed onto the top of stack is a zero to indicate no more
                                     1107 3745      ; segments. R1 is pointing to the first byte of user data.
                                     1107 3746
52 OC84 C4 9E 1107 3747      MOVAB    PDT$L_PQ_ALGNPG(R4),R2      ; R2 => Aligned page.
                                     18 B5 17 110C 3748      JMP     @CDRP$L_SAVD_RTN(R5)      ; Return to caller leaving data on stack
                                     110F 3749
                                     110F 3750      ;+
                                     110F 3751      ; SETUP_COPY_SEG2 - Routine to setup copy of the second segment of user data
                                     110F 3752      ; from (to) the user buffer to (from) the aligned page.
                                     110F 3753
                                     110F 3754      ; Inputs:
                                     110F 3755      ; PDT$L_PQ_USRPTE = System address of PTE that describes previous page
                                     110F 3756      ; of user buffer
                                     110F 3757      ; PDT$L_PQ_SVPSTE = Slot in System Page Table to map user buffer
                                     110F 3758      ; PDT$L_PQ_UBFSVA = System Space address that corresponds to this slot
                                     110F 3759
                                     110F 3760      ; Outputs:
                                     110F 3761      ; PDT$L_PQ_USRPTE = PDT$L_PQ_USRPTE + 4
                                     110F 3762      ; R1 => Byte zero of page that corresponds to slot defined by
                                     110F 3763      ; PDT$L_PQ_SVPSTE.
                                     110F 3764
                                     110F 3765      ; Note: R2 must be preserved.
                                     110F 3766      ; -
                                     110F 3767
                                     110F 3768      SETUP_COPY_SEG2:
                                     110F 3769
OC80 C4 04 C0 110F 3770      ADDL     #4,PDT$L_PQ_USRPTE(R4)      ; Point to next user PTE.
                                     15 00 EF 1114 3771      EXTZV    #PTE$V_PFN,#PTE$S_PFN,-      ; Construct a PTE with the proper PFN,
OC80 D4      @PDT$L_PQ_USRPTE(R4),-      ; protection=KW, and own=K. Here we
OCAC D4      @PDT$L_PQ_SVPSTE(R4)      ; move the PFN.
                                     C8 111D 3774      BISL     #PTE$M_VALID!-      ; And here we set VALID, the protection
                                     111E 3775      ; PTE$C_KW!-      ; and the ownership and thereby map
                                     111E 3776      ; PTE$C_KOWN,-      ; the next page of the user buffer
OCAC D4 90000000 8F 111E 3777      @PDT$L_PQ_SVPSTE(R4)      ; into System Space.
51 OCA8 C4 D0 1126 3778      MOVL     PDT$L_PQ_UBFSVA(R4),R1      ; R1 => Rest of user data.
                                     112B 3779      INVALID  R1      ; Invalidate virtual address.
                                     05 112E 3780      RSB     ; Return to caller.

```


PUDRIVER
V04-000

I 14

16-SEP-1984 01:05:05 VAX/VMS Macro V04-00
5-SEP-1984 00:17:10 [DRIVER.SRC]PUDRIVER.MAR;1

Page 84
(5)

+ FPC\$MAPIRP_BDA, Map a buffer for BDA

```
112F 3782 .SBTTL +
112F 3783
112F 3784 ;+
112F 3785 ; Place holder.
112F 3786 ; -
112F 3787
05 112F 3788 FPC$MAPIRP_BDA::
112F 3789 RSB
```

FPC\$MAPIRP_BDA, Map a buffer for BDA

+ FPC\$UNMAP, Release mapping resources

```

1130 3791      .SBTTL +      FPC$UNMAP,      Release mapping resources
1130 3792
1130 3793      :+
1130 3794      : FPC$UNMAP
1130 3795      :
1130 3796      : Inputs:
1130 3797      : R5 => CDRP
1130 3798      : CDRP$L_UBARSRCE contains accounting of allocated resources.
1130 3799      : CDRP$L_LBUFH_AD => local buffer handle.
1130 3800      :
1130 3801      : Outputs:
1130 3802      : Resources returned
1130 3803      : CDRP$L_LBUFH_AD cleared
1130 3804      : R0-R2 destroyed
1130 3805      : Other registers preserved
1130 3806      :-
1130 3807
1130 3808      FPC$UNMAP::
1130 3809
1130 3810      PUSHL R3 ; Save register.
1130 3811      PUSHL R5 ; Save another.
1130 3812      MOVB CDRP$L_UBARSRCE+UBMDSB_DATAPATH(R5),R0 ; R0 contains datapath
1130 3813      ; used in transfer.
1130 3814      MOVL PDT$L_UCB0(R4),R5 ; R5 => UCB.
1130 3815      MOVL UCB$L_CRB(R5),R3 ; R3 => CRB.
1130 3816      MOVB R0,CRB$L_INTD+VECSB_DATAPATH(R3); Set path number for purge.
1130 3817      JSB G^IOC$PURGDATAP ; Purge specified data path.
1130 3818      CLRB CRB$L_INTD+VECSB_DATAPATH(R3) ; Clear datapath # from CRB.
1130 3819      POPL R5 ; Restore register.
1130 3820      POPL R3 ; Restore another.
1130 3821
1130 3822      JSB G^IOC$RELMAPUDA ; Return allocated map registers.
1130 3823
1130 3824      JSB @RELDATAPATH_TV ; And also the datapath.
1130 3825
1130 3826      CLRL CDRP$L_LBUFH_AD(R5) ; Prevent spurious deallocates.
1130 3827      RSB ; Return to caller.
1130 3828

```


+ FPC\$UNMAP_UV1, Release mapping resource

```
1162 3830 .SBTTL + FPC$UNMAP_UV1, Release mapping resources
1162 3831
1162 3832 :+
1162 3833 : FPC$UNMAP_UV1
1162 3834 :
1162 3835 : Inputs:
1162 3836 : R5 => CDRP
1162 3837 : CDRP$L_UBARSRC contains accounting of allocated resources.
1162 3838 : CDRP$L_LBUFH_AD => local buffer handle.
1162 3839 :
1162 3840 : Outputs:
1162 3841 : Resources returned
1162 3842 : CDRP$L_LBUFH_AD cleared
1162 3843 : R0-R2 destroyed
1162 3844 : Other registers preserved
1162 3845 :-
1162 3846
1162 3847 FPC$UNMAP_UV1::
1162 3848
1162 3849 BLBS CDRP$W_BOFF(R5),- ; Branch around if Not word aligned.
1165 3850 UNMAP_UNALIGN
1166 3851 UNMAP_ODD: ; Label to allow branch back.
1166 3852
1166 3853 JSB G^IOC$RELMAPUDA ; Return allocated map registers.
116C 3854
116C 3855 CLRL CDRP$L_LBUFH_AD(R5) ; Prevent spurious deallocates.
116F 3856 RSB ; Return to caller.
1170 3857
1170 3858 UNMAP_UNALIGN:
1170 3859
1170 3860 ; Here after successfully using the aligned Page, if the user operation
1170 3861 ; is a READ, we copy the user's data from the aligned page. If the user
1170 3862 ; operation is a WRITE, we branch around the copy and merely deallocate
1170 3863 ; the aligned Page. The data from a WRITE is copied from the user
1170 3864 ; buffer to the aligned Page when the aligned page is allocated.
1170 3865
1170 3866 MOVL PDT$L_UCB0(R4),R0 ; R0 => port UCB.
1175 3867 BBS #PS1 V_OD,- ; If controller supports odd addresses,
1177 3868 UCBS$W_PORTSTEP1(R0),- ; then branch back in line.
117A 3869 UNMAP_ODD
117B 3870 BBC #IRP$V_FUNC,- ; If a WRITE, branch around.
117D 3871 CDRP$W_STS(R5),20$
1180 3872 BSBW SETUP_COPY_SEG1 ; Call to setup copy of user data.
1183 3873 ; This is a complex routine that
1183 3874 ; returns: R0 = # bytes in first
1183 3875 ; segment, R1 => user data,
1183 3876 ; R2 => aligned page, (SP) = # bytes
1183 3877 ; in second segment (or zero if none),
1183 3878 ; 4(SP) (valid only if (SP) non-zero)
1183 3879 ; zero.
1183 3880 10$:
1183 3881 MOVB (R2)+,(R1)+ ; Copy a byte.
1186 3882 SOBGTR R0,10$ ; Loop thru entire segment.
1189 3883 POPL R0 ; Get length of next segment (if any).
118C 3884 BEQL 20$ ; EQL implies no more.
118E 3885 BSBW SETUP_COPY_SEG2 ; Call to setup copy of second segment.
1191 3886 BRB 10$ ; Branch back to loop.
```

DO A5 E8
OA
00000000'GF 16
2C A5 D4
05
50 00DC C4 D0
07 E0
00AC C0
EB
01 E1
13 CA A5
FF23 30
81 82 90
FA 50 F5
50 8ED0
05 13
FF7E 30
F0 11

+ FPC\$UNMAP_UV1, Release mapping resource

	OCA0	C4	D4	1193	3887	20\$:			
	2C	A5	D4	1193	3888		CLRL	PDT\$ _L PQ POWNER(R4)	; Release Aligned Page.
50	0C98	D4	0F	1197	3889		CLRL	CDRP\$ _L LBUFH AD(R5)	; Prevent spurious deallocates.
		1B	1D	119A	3890		REMQUE	@PDT\$ _L PQ_PGQFL(R4),R0	; R0 => Waiting CDRP (if any).
	7E	53	7D	119F	3891		BVS	30\$; VS implies no waiters.
		55	DD	11A1	3892		MOVQ	R3,-(SP)	; Save registers before resuming waiter.
				11A4	3893		PUSHL	R5	
	55	50	D0	11A6	3894				
53	10	A5	7D	11A6	3895		MOVL	R0,R5	; R5 => Waiter's CDRP.
	FE9B		30	11A9	3896		MOVQ	CDRP\$ _L FR3(R5),R3	; Restore his registers.
				11AD	3897		BSBW	MAP_UNALIGN	; Call to allocate and copy data.
				11B0	3898				; Note allocation MUST succeed.
00000000	'GF		16	11B0	3899		JSB	G^SCS\$RESUMEWAITR	; Resume waiting thread and any backed
				11B6	3900				; up IRP's.
				11B6	3901				
	55	8E	D0	11B6	3902		POPL	R5	; Restore our original registers.
53	8E		7D	11B9	3903		MOVQ	(SP)+,R3	
				11BC	3904	30\$:			
			05	11BC	3905		RSB		; Return to caller.

PUDRIVER
V04-000

M 14

+ FPC\$UNMAP_BDA, Release mapping resourc 16-SEP-1984 01:05:05 VAX/VMS Macro V04-00 Page 88
5-SEP-1984 00:17:10 [DRIVER.SRC]PUDRIVER.MAR;1 (5)

```
11BD 3907 .SBTTL + FPC$UNMAP_BDA, Release mapping resources BDA
11BD 3908
11BD 3909 ::+
11BD 3910 :: Place holder.
11BD 3911 ::-
11BD 3912
05 11BD 3913 FPC$UNMAP_BDA::
11BD 3914 RSB
```

```

11BE 3916      .SBTTL  INTERNAL SUBROUTINES
11BE 3917      .SBTTL  +      POLL_CMDRING
11BE 3918
11BE 3919      +
11BE 3920      POLL_CMDRING is called to poll the command ring and reclaim any slots (and
11BE 3921      the buffers pointed to by them) that have been released back to the host by
11BE 3922      the port. POLL_CMDRING makes use of some PDT fields:
11BE 3923
11BE 3924      PDT$B_CRINGCNT - which maintains the count of how many as yet
11BE 3925      unreclaimed slots the host has sent to the port.
11BE 3926
11BE 3927      PDT$B_CPOLLINX - whose low order UDASK_RINGEXP bits contain the index
11BE 3928      of the command ring slot which we should poll next.
11BE 3929
11BE 3930      PDT$B_CRCONTENT - an array of UDASK_RINGSIZE bytes, each of which
11BE 3931      maintains the index of the buffer currently pointed
11BE 3932      at by the corresponding ring slot.
11BE 3933
11BE 3934      PDT$L_BDTABLE - an array of UDASK_RINGSIZE longwords, each of which
11BE 3935      point to the buffer corresponding to the index of the
11BE 3936      longword.
11BE 3937
11BE 3938      POLL_CMDRING polls ring slots to see if the port has released them until
11BE 3939      either of the following two conditions obtain: PDT$B_CRINGCNT goes to zero,
11BE 3940      indicating that all command ring slots sent to the port have been reclaimed;
11BE 3941      or upon polling a slot we come upon one that has NOT been released as yet.
11BE 3942      Since slots are released in sequence, this means that we should cease
11BE 3943      polling.
11BE 3944
11BE 3945      POLL_CMDRING always polls the slot selected by PDT$B_CPOLLINX. The low order
11BE 3946      bits of this field are extracted and used as an index into the two arrays
11BE 3947      mentioned above.
11BE 3948
11BE 3949      A slot is polled by testing its high order bit. A zero bit indicates that
11BE 3950      the slot has been released to the host. Upon finding a released slot,
11BE 3951      POLL_CMDRING reclaims it and the buffer pointed to it by:
11BE 3952
11BE 3953      1. The index of the buffer is obtained from the element of the
11BE 3954      PDT$B_CRCONTENT array corresponding to the current ring slot.
11BE 3955
11BE 3956      2. A pointer to the buffer is obtained from the PDT$L_BDTABLE array.
11BE 3957
11BE 3958      3. PDT$B_CPOLLINX is incremented so that the next poll will use the
11BE 3959      next slot in the command ring.
11BE 3960
11BE 3961      4. PDT$B_CRINGCNT is decremented to show one less unreclaimed slot.
11BE 3962
11BE 3963      5. If any buffers are queued waiting for available ring slots on
11BE 3964      PDT$L_PU_SNDQFL, the first one is removed from the Q and
11BE 3965      inserted into the command ring by calling internal subroutine
11BE 3966      INSERT_IN_CRING.
11BE 3967
11BE 3968      6. The now free buffer is put onto the response ring or the free Q,
11BE 3969      whichever is appropriate, by calling internal subroutine,
11BE 3970      Q_DEALLOC_BUF.
11BE 3971
11BE 3972      7. Finally we branch back to the beginning of POLL_CMDRING to poll

```


+ POLL_CMDRING

```
11BE 3973 : again.
11BE 3974 :
11BE 3975 : Inputs:
11BE 3976 :
11BE 3977 : R4 -Addr of PDT
11BE 3978 :
11BE 3979 : Outputs:
11BE 3980 :
11BE 3981 : R0-R2 -destroyed
11BE 3982 : Other registers -preserved
11BE 3983 :-
11BE 3984 :
11BE 3985 POLL_CMDRING:
11BE 3986 :
0122 C4 95 11BE 3987 TSTB PDT$B_CRINGCNT(R4) ; See if any slots in use on command
36 13 11C2 3988 BEQL 20$ ; ring. EQL implies NO.
11C4 3989 :
52 04 00 EF 11C4 3990 EXTZV #0,#UDASK_RINGEXP,- ; Extract ring index of slot of
0121 C4 11C7 3991 PDT$B_CPOCLINX(R4),R2 ; where to begin polling.
11CB 3992 :
0248 C442 D5 11CB 3993 TSTL PDT$L_CMDRING(R4)[R2] ; Has controller released this slot?
28 19 11D0 3994 BLSS 20$ ; LSS implies NO.
11D2 3995 :
52 012C C442 9A 11D2 3996 MOVZBL PDT$B_CRCONTENT(R4)[R2],R2 ; R2 = index of buffer pointed
11D8 3997 ; at by this slot.
52 014C C442 D0 11D8 3998 MOVL PDT$L_BDTABLE(R4)[R2],R2 ; R2 => buffer header.
11DE 3999 :
52 DD 11DE 4000 PUSHL R2 ; Remember R2 => free buffer.
11E0 4001 :
0121 C4 96 11E0 4002 INCB PDT$B_CPOCLINX(R4) ; Bump polling index.
0122 C4 97 11E4 4003 DECB PDT$B_CRINGCNT(R4) ; One less used command ring slot.
11E8 4004 :
52 0110 D4 OF 11E8 4005 REMQUE @PDT$L_PU_SNDQFL(R4),R2 ; R2 => buffer (if any) to be inserted
03 1D 11ED 4006 BVS 10$ ; in command ring. VS implies NONE.
FDAC 30 11EF 4007 BSBW INSERT_IN_CRING ; Call to insert R2 => buffer in ring.
11F2 4008 10$:
52 8ED0 11F2 4009 POPL R2 ; Restore R2 => free buffer.
FCA5 30 11F5 4010 BSBW Q_DEALLOC_BUF ; Call to put free buffer onto response
11F8 4011 ; ring or onto free Q, whichever is
11F8 4012 ; appropriate.
C4 11 11F8 4013 BRB POLL_CMDRING ; Branch back to reclaim more buffers.
11FA 4014 20$:
05 11FA 4015 RSB ; Return to caller.
```


- STATE_ERR, RETURN CDT STATE ERROR

11FB 4017 .SBTTL - STATE_ERR, RETURN CDT STATE ERROR
11FB 4018 .SBTTL - TO SYSAP
11FB 4019
11FB 4020 ;+
11FB 4021 ; Set error status code and return to caller.
11FB 4022 ;-
11FB 4023

50 2154 8F

3C
05

11FB 4024 STATE_ERR:
11FB 4025

11FB 4026 MOVZWL #SS\$_ILLCDTST,R0
1200 4027 RSB
1201 4028
1201 4029

; Status = illegal CDT state
; Return to SYSAP

INTERRUPT SERVICING

```
1201 4031 .SBTTL INTERRUPT SERVICING
1201 4032 .SBTTL + PUSINT - Interrupt service routine
1201 4033 :
1201 4034 : Inputs:
1201 4035 : 00(SP) - Pointer to IDB
1201 4036 : 04(SP) - SAVED R0
1201 4037 : 08(SP) - SAVED R1
1201 4038 : 12(SP) - SAVED R2
1201 4039 : 16(SP) - SAVED R3
1201 4040 : 20(SP) - SAVED R4
1201 4041 : 24(SP) - SAVED R5
1201 4042 : 28(SP) - PC AT THE TIME OF THE INTERRUPT
1201 4043 : 32(SP) - PSL AT THE TIME OF THE INTERRUPT
1201 4044 :
1201 4045 :
1201 4046 PUSINT::
55 53 9E D0 1201 4047 MOVL @ (SP)+,R3 ; Get address of IDB
54 04 A3 D0 1204 4048 MOVL IDB$$_OWNER(R3),R5 ; Get address of UCB
0084 C5 D0 1208 4049 MOVL UCB$$_PDT(R5),R4 ; Get address of PDT
53 63 D0 120D 4050 MOVL IDB$$_CSR(R3),R3 ; R3 => UDA CSR.
02 A3 B0 1210 4051 MOVW UDASA(R3),-
00AA C5 1213 4052 UCB$$_UDASA(R5) ; Save error status in UCB.
1216 4053
50 0203 C4 90 1216 4054 MOVB PDT$$_PURGEDP(R4),R0 ; Fetch Purge data path number
28 13 121B 4055 BEQL 10$ ; Br if none
0203 C4 94 121D 4056 CLRB PDT$$_PURGEDP(R4) ; Clear it
51 24 A5 D0 1221 4057 MOVL UCB$$_CRB(R5),R1 ; R1 => CRB to setup for purge
52 37 A1 9A 1225 4058 MOVZBL CRB$$_INTD+VEC$$_DATAPATH(R1),R2 ; Save datapath #, in case we
1229 4059 ; interrupted purge in UNMAP.
7E 51 7D 1229 4060 MOVQ R1,-(SP) ; Push R1=>CRB, R2=datapath #.
37 A1 50 90 122C 4061 MOVB R0,CRB$$_INTD+VEC$$_DATAPATH(R1) ; Set path number for purge
00000000 GF 16 1230 4062 JSB G*IOC$$_PURGDATAP ; Purge specified data path
53 0100 C4 D0 1236 4063 MOVL PDT$$_PU_CSR(R4),R3 ; Refresh R3=>CSR after purge
02 A3 B4 123B 4064 CLRW UDASA(R3) ; Acknowledge purge interrupt
51 8E 7D 123E 4065 MOVQ (SP)+,R1 ; Pop R1=>CRB, R2=Datapath #.
37 A1 52 90 1241 4066 MOVB R2,CRB$$_INTD+VEC$$_DATAPATH(R1) ; Restore old contents of
1245 4067 ; datapath # to CRB.
1245 4068 10$:
09 64 A5 E5 1245 4069 BBCC #UCB$$_INT,- ; See if we are awaiting interrupt. If
1247 4070 UCB$$_STS(R5),20$ ; so then we are in hardware init. If
124A 4071 ; not, branch around.
53 10 A5 7D 124A 4072 MOVQ UCB$$_FR3(R5),R3 ; Restore initialization thread
0C B5 16 124E 4073 JSB @UCB$$_FPC(R5) ; context.
07 11 1251 4074 BRB 30$ ; And branch around to dismiss interrupt.
1253 4075 20$:
1253 4076
02 68 A5 E2 1253 4077 BBSS #UCB$$_PU FRKBSY,- ; Appropriate UCB fork block, or else
1255 4078 UCB$$_DEVSTS(R5),30$ ; branch to dismiss interrupt
03 10 1258 4079 BSBB POLL_RSPRING ; If we were successful in appropriating
125A 4080 ; the fork block, BSBB to poll the
125A 4081 ; response ring.
3F BA 125A 4082 30$:
02 125A 4083 POPR #*M<R0,R1,R2,R3,R4,R5> ; Restore registers.
125C 4084 REI ; And dismiss interrupt.
```


+ POLL_RSPRING

```
125D 4086 .SBTTL + POLL_RSPRING
125D 4087
125D 4088
125D 4089 :+
125D 4090 : POLL_RSPRING is called from the interrupt service routine at device IPL and
125D 4091 : with the UCB fork allocated to this thread (i.e. UCBSM_PU_FRKBSY bit on in
125D 4092 : UCBSW_DEVSTS). POLL_RSPRING first IOFORK's on the UCB so that the interrupt
125D 4093 : can be dismissed and then after resuming execution at fork IPL, it frees up
125D 4094 : the UCB fork block (i.e. clears UCBSM_PU_FRKBSY) and tests whether a power
125D 4095 : failure has occurred recently. If so, then we merely branch out of the flow
125D 4096 : here to begin a thread at POST_POWER_FORK, that causes all CONNECTION's to
125D 4097 : resynchronize.
125D 4098 :
125D 4099 : If we remain here (normal case) we traverse the response ring looking for
125D 4100 : buffers that have been released to the host (us), and upon finding one
125D 4101 : we determine whether it is a SEQUENCED MESSAGE or a DATAGRAM and we call
125D 4102 : the appropriate entrypoint in the SYSAP for the CONNECTION over which the
125D 4103 : message was received.
125D 4104 :
125D 4105 : Inputs:
125D 4106 : R4 -Addr of PDT
125D 4107 : R5 -Addr of UCB
125D 4108 :
125D 4109 : UCB fork block allocated to this thread
125D 4110 :
125D 4111 : Outputs:
125D 4112 :
125D 4113 : Response ring polled and SYSAPs called
125D 4114 :
125D 4115 : .enabl lsb
125D 4116 POLL_RSPRING:
125D 4117
125D 4118 IOFORK ; Lower IPL.
125D 4119 BICW #UCBSM_PU_FRKBSY,- ; Allow fork block to be re-used.
125D 4120 UCBSW_DEVSTS(R5) ; See if we got a fatal error.
125D 4121 TSTW UCBSW_UDASA(R5) ; LSS means fatal error.
125D 4122 BLSS 5$ ; If set, then MRESET request received
125D 4123 BBS #UCBSV_PU_MRESET,- ; while we were forked, so go to do it.
125D 4124 UCBSW_DEVSTS(R5),8$ ; See if POWERFAIL occurred while
125D 4125 BBC #UCBSV_POWER,- ; UCB fork block was busy. Branch if not.
125D 4126 UCBSW_STS(R5),10$ ; Branch around if YES powerfail.
125D 4127 BRB 8$
125D 4128 5$:
125D 4129 MOVW #UDASA_ATTNCODE,- ; Indicate what kind of error log record
125D 4130 UCBSW_ATTNCODE(R5) ; we are about to create.
125D 4131 JSB G^ERL$DEVICEATTN ; Call to create error log record of INIT.
125D 4132 8$:
125D 4133 BRW POST_POWER_FORK ; If POWERFAIL, get out of here.
125D 4134 10$:
125D 4135 BBCC #0,PDT$W_CMDINT(R4),20$ ; Branch if command ring NOT UNfull.
125D 4136 BSBW POLL_CMDRING ; Reclaim free space in command ring.
125D 4137 20$:
125D 4138 CLRW PDT$W_RSPINT(R4) ; Always clear response interrupt
125D 4139 ; indication since we always poll
125D 4140 ; response ring on interrupt.
125D 4141 30$:
125D 4142 TSTB PDT$B_RRINGCNT(R4) ; See if response ring has anything to
```

02 AA 1263 4119
68 A5 1265 4120
00AA C5 B5 1267 4121
OC 19 1268 4122
04 E0 126D 4123
12 68 A5 126F 4124
05 E1 1272 4125
10 64 A5 1274 4126
0B 11 1277 4127
03 B0 1279 4128
00A8 C5 1279 4129
00000000 GF 16 127E 4131
F03C 31 1284 4132
03 0204 C4 00 E5 1287 4133
FF2E 30 1287 4134
0206 C4 B4 128D 4135
1290 4136
1290 4137
1294 4138
1294 4139
1294 4140
0125 C4 95 1294 4141


```
+ POLL_RSPRING

01 12 1298 4143 BNEQ 50$ ; poll. NEQ implies YES.
    129A 4144 40$: RSB ; If no more, kill this thread.
05 129A 4145 50$:
    129B 4146
50 04 00 EF 129B 4147 EXTZV #0,#UDASK_RINGEXP,- ; Extract index mod sing size leaving
    0124 C4 129E 4148 PDTSB_RPOLINX(R4),R0 ; R0 = index of slot to poll.
    0208 C440 D5 12A2 4149 TSTL PDTSB_RSPRING(R4)[R0] ; See if slot released to host.
    F1 19 12A7 4150 BLSS 40$ ; LSS => slot still owned by controller.
    12A9 4151
    0124 C4 96 12A9 4152 INCB PDTSB_RPOLINX(R4) ; Bump response poll index.
    0125 C4 97 12AD 4153 DECB PDTSB_RRINGCNT(R4) ; Decr # slots passed to controller.
50 013C C440 9A 12B1 4154 MOVZBL PDTSB_RRCONTENT(R4)[R0],R0 ; R0 = index of buffer with response.
53 014C C440 D0 12B7 4155 MOVL PDTSB_BDTABLE(R4)[R0],R3 ; R3 => buffer with response.
    12BD 4156
    12BD 4157 ENABLE_RESPONSE_START:
52 0108 D4 0F 12BD 4158 REMQUE @PDTSB_PU_FQFL(R4),R2 ; R2 => free buffer. NOTE this instruction
    12C2 4159 ; is copied below in the flow that gets
    12C2 4160 ; executed if tracing is enabled. Any
    12C2 4161 ; edit to this instruction should also
    12C2 4162 ; done to its copy.
    12C2 4163 ENABLE_RESPONSE_END:
    0E 1C 12C2 4164 BVC 55$ ; VC implies R2 => buffer.
    FEF7 30 12C4 4165 BSBW POLL_CMDRING ; If no buffers, reclaim some.
    10 91 12C7 4166 CMPB #UDASK_RINGSIZE,- ; For debugging, assure that we indeed
    0125 C4 12C9 4167 PDTSB_RRINGCNT(R4) ; have a full response ring.
    07 13 12CC 4168 BEQL 57$ ; EQL implies full response ring.
    12CE 4169 BUG_CHECK UDAPOINT,FATAL ; Else bug check for now.
    12D2 4170 55$:
    FBCF 30 12D2 4171 BSBW INSERT_IN_RRING ; Else insert free buffer in response
    12D5 4172 ; ring.
    12D5 4173
50 53 D0 12D5 4174 57$: MOVL R3,R0 ; R0 => buffer with response.
    12D8 4175
    12D8 4176 : MOVZBL UDABSB_CONID(R0),R3 ; R3 = connection ID of message.
    12D8 4177 : Here we assume that VC0 is DISK MSCP, VC1 is TAPE MSCP, VC2 is DUP and
    12D8 4178 : VC255 (low order bits both on) so that VC255 maps to VC3
    02 00 EF 12D8 4179 EXTZV #0,#2,- ; Low order 2 bits select connection.
53 53 13 A0 12DB 4180 UDABSB_CONID(R0),R3 ; R3 = connection ID of message.
    00E4 C443 D0 12DE 4181 MOVL PDTSB_PU_CDTARY(R4)[R3],R3 ; R3 => CDT.
    12E4 4182
    00 EF 12E4 4183 EXTZV #UDABSV_CREDITS,- ; Extract credits returned by message.
    04 12E6 4184 #UDABSS_CREDITS,-
51 12 A0 12E7 4185 UDABSB_CREDTYPE(R0),R1
40 A3 51 A0 12EA 4186 ADDW R1,CDTSW_SEND(R3) ; Add in new credits.
    12EE 4187
52 14 A0 9E 12EE 4188 MOVAB UDABST_TEXT(R0),R2 ; R2 => application area of message.
51 10 A0 3C 12F2 4189 MOVZWL UDABSW_MSG_LEN(R0),R1 ; Pickup length of datagram or message.
    7E 53 7D 12F6 4190 MOVQ R3,-(SP) ; Save R3=>CDT and R4=>PDT before
    12F9 4191 ; dispatching.
    04 EF 12F9 4192 EXTZV #UDABSV_MSGTYPE,- ; Extract type of message.
    04 12FB 4193 #UDABSS_MSGTYPE,-
55 12 A0 12FC 4194 UDABSB_CREDTYPE(R0),R5
    12FF 4195 ASSUME UDASK_SEQMSGTYP EQ 0
    23 12 12FF 4196 BNEQ 90$ ; NEQ means NOT sequenced message.
    00 B3 16 1301 4197 JSB @CDTSB_MSGINPUT(R3) ; Call sequenced message handler
    1304 4198 ; passing R2 => message text.
    1304 4199 85$:
```



```

+ POLL_RSPRING
53 8E 7D 1304 4200 MOVQ (SP)+,R3 ; Restore R3=>CDT and R4=>PDT after
1307 4201 ; dispatching.
1307 4202 87$: TSTW CDT$W_SEND(R3) ; See if we are positive here.
40 A3 B5 1307 4203 BLEQ 89$ ; LEQ means no more credits left.
15 15 130A 4204 $RESUME FP - ; Resume anyone waiting for credits
130C 4205 @CDT$L_CRWAITQFL(R3),-
130C 4206 QEMPTY=89$ ; Where to go if no waiters.
E6 11 131F 4208 BRB 87$ ; Go back and try to resume more.
FF70 31 1321 4209 89$: BRW 30$ ; Go back to test for more responses.
1324 4211 90$: CMPL #UDASK_DGTYPE,R5 ; See if Datagram message.
55 01 D1 1324 4212 BNEQ 100$ ; NEQ means test for something else.
05 12 1327 4213 JSB @CDT$L_DGINPUT(R3) ; Call datagram handler passing
04 B3 16 1329 4214 ; R2 => datagram, with R1 = length.
132C 4215 BRB 85$ ; Go back to test for more responses.
D6 11 132E 4216 100$: CMPL #UDASK_CREDTYPE,R5 ; See if CREDIT message.
55 02 D1 132E 4218 BNEQ 110$ ; NEQ means test for something else.
05 12 1331 4219 BSBW NULL_MSG_INPUT ; Call Null Message handler to dispose
EDA6 30 1333 4220 ; of and recycle buffer.
1336 4221 BRB 85$ ; Go back to test for more responses.
CC 11 1338 4222 110$: CMPL #UDASK_MAINTTYPE,R5 ; See if MAINTENANCE message.
55 0F D1 1338 4224 BNEQ 120$ ; NEQ means test for something else.
0F 12 133B 4225 EXTZV #UDAB$V_CREDITS,- ; The Credit Field of Maintenance
00 EF 133D 4226 #UDAB$S_CREDITS,- ; messages is to be ignored. So we
04 133F 4227 UDAB$B_CREDTYPE(R0),R0 ; again extract credits field and
50 12 A0 1340 4228 SUBW R0,CDT$W_SEND(R3) ; Subtract out credits added in above.
40 A3 50 A2 1343 4229 BSBW NULL_DG_INPUT ; Call Null Datagram handler to log
ED96 30 1347 4231 ; message and then recycle buffer.
134A 4232 BRB 85$ ; Go back to test for more responses.
B8 11 134A 4233 120$: BUG_CHECK UDAPORT,FATAL
134C 4234 ENABLE_RESPONSE_CODE: ; If we enable tracing, the driver will
134C 4235 ; dynamically patch location
1350 4236 ; ENABLE_RESPONSE_START to BRW here.
1350 4237 BSBW TRACE_RESPONSE ; Copy response buffer to trace table.
1350 4238
1350 4239
F12B 30 1353 4241 REMQUE @PDT$L_PU_FQFL(R4),R2 ; R2 => free buffer.
52 0108 D4 0F 1353 4242 BRW ENABLE_RESPONSE_END ; Branch back to normal stream.
FF67 31 1358 4243
135B 4244
00000090 135B 4245 ENABLE_RESPONSE_OFFSET=ENABLE_RESPONSE_CODE-ENABLE_RESPONSE_START-3
135B 4246 .dsabl 1sb

```


+ PUSSA_POLL

```

135B 4248 .SBTTL + PUSSA_POLL
135B 4249
135B 4250 :+
135B 4251 : Routine periodically called by CRB wakeup mechanism to see if the SA register
135B 4252 : indicates that this port has had an uncorrectable error. If so the port is
135B 4253 : re-initialized in order to bring it back.
135B 4254 :
135B 4255 : Inputs:
135B 4256 : R3 => CRB.
135B 4257 :
135B 4258 : Outputs:
135B 4259 : All registers, R0-R5 are modified.
135B 4260
135B 4261 PUSSA_POLL:
135B 4262
54 10 A3 D0 135B 4263 MOVL CRB$L_AUXSTRUC(R3),R4 ; R4 => PDT.
55 00DC C4 D0 135F 4264 MOVL PDT$L_UCB0(R4),R5 ; R5 => UCB.
50 0100 C4 D0 1364 4265 MOVL PDT$L_PU_CSR(R4),R0 ; R0 => Port CSR.
    02 A0 B0 1369 4266 MOVW UDASA7R0,- ; Retrieve SA register
    00AA C5 136C 4267 UCB$W_UDASA(R5) ; and check for error.
    OF 19 136F 4268 BLSS 10$ ; LSS means YES, error.
    OF C1 1371 4269
00000000'GF 18 A3 1371 4270 ADDL3 #SA_POLL_INTVAL,- ; Here we had no error, so we simply
    DE AF 9E 1373 4271 G^EXESGL_ABSTIM,- ; re-establish SA polling interval.
    1C A3 1378 4272 CRB$L_DUE TIME(R3)
    137A 4273 MOVAB PUSSA_POLL,- ; And re-establish wakeup routine.
    137D 4274 CRB$L_TOUTROUT(R3)
    05 137F 4275 RSB ; And simply return to caller.
1380 4276 10$: ; Here we had an SA error.
1380 4277 SETIPL #IPL$ _SCS ; Lower IPL for processing.
    03 B0 1383 4278 MOVW #UDASA_ATTNCODE,- ; Indicate what kind of errorlog
    00A8 C5 1385 4279 UCB$W_ATTNCODE(R5) ; record we are about to create.
00000000'GF 16 1388 4280 JSB G^ERL$DEVICEATTN ; Call to create error log record of SA
    00AA C5 B4 138E 4281 ; error.
    EF2E 31 138E 4282 CLRW UCB$W_UDASA(R5) ; Clear so that we do not report error
    1392 4283 ; redundantly.
    1392 4284 BRW POST_POWER_FORK ; Branch to begin re-init of port.
    1395 4285
    1395 4286 PU$END:
    1395 4287 .END

```


PUDRIVER
Symbol table

I 15

16-SEP-1984 01:05:05 VAX/VMS Macro V04-00
5-SEP-1984 00:17:10 [DRIVER.SRC]PUDRIVER.MAR;1

Page 97
(5)

\$\$\$	= 00000020	R	02
\$\$\$ENTRYNUM	= 0000001E		
\$\$\$PREV	= 00000080		
\$\$BASE	= 00000001		
\$\$DISPL	= 00000008		
\$\$GENSW	= 00000001		
\$\$HIGH	= 00000007		
\$\$LIMIT	= 00000006		
\$\$LOW	= 00000001		
\$\$MNSW	= 00000001		
\$\$MXSW	= 00000001		
\$\$OP	= 00000002		
ADPSL_CSR	= 00000000		
ADPSL_VECTOR	= 00000010		
ADPSW_ADPTYPE	= 0000000E		
ADPSW_TR	= 0000000C		
AFTER_RINGBASE	= 00000787	R	03
ALLOC_DELTA	= 00000001		
ALLOC_PDT_8SS	= 0000031D	R	03
ALLOC_PDT_NOTUV1	= 0000032F	R	03
ALLOC_PDT_UV1	= 0000033D	R	03
ALLOC_POOC	= 00000C8C	R	03
ATS_UBA	= 00000001		
BRW_OPCODE	= 00000031		
BUGS_UDAPORT	*****	X	03
BUGS_UNSUPRTCPU	*****	X	03
BUILD_PB_SB	= 00000BBF	R	03
BUILD_PDT	= 00000303	R	03
CDRPSL_BCNT	= FFFFFFFD2		
CDRPSL_CDT	= 00000024		
CDRPSL_FPC	= 0000000C		
CDRPSL_FQFL	= 00000000		
CDRPSL_FR3	= 00000010		
CDRPSL_LBUFH_AD	= 0000002C		
CDRPSL_MSG_BUF	= 0000001C		
CDRPSL_RWCPT	= 00000028		
CDRPSL_SAVD_RTN	= 00000018		
CDRPSL_SVAPTE	= FFFFFFFCC		
CDRPSL_UBARSCE	= 0000003C		
CDRPSW_BOFF	= FFFFFFFD0		
CDRPSW_STS	= FFFFFFFCA		
CDTSC_CLOSED	= 00000000		
CDTSC_OPEN	= 00000002		
CDTSK_LENGTH	= 000000A0		
CDTSL_CRWAITQBL	= 0000003C		
CDTSL_CRWAITQFL	= 00000038		
CDTSL_DGINPUT	= 00000004		
CDTSL_ERRADDR	= 0000000C		
CDTSL_MSGINPUT	= 00000000		
CDTSL_RCONID	= 00000014		
CDTSL_RPROCNAM	= 00000050		
CDTSW_REASON	= 00000026		
CDTSW_SEND	= 00000040		
CDTSW_STATE	= 00000028		
CNTRLTYP_ARRAY	= 000007C3	R	03
COMAREA_BOFF	= 00000000		
COMPLETE_IO	= 00000085	R	03

COM_INIT_UDA_BUFS	= 00000512	R	03
CONREJ	= 00000D6F	R	03
CONVCO	= 00000DA5	R	03
CONVC1	= 00000D96	R	03
CONVC2	= 00000D87	R	03
CON_COMMON	= 00000DB1	R	03
CON_NO_LISTEN	= 00000D7B	R	03
CON_NO_NODE	= 00000D63	R	03
CRBSL_AUXSTRUC	= 00000010		
CRBSL_DUE TIME	= 00000018		
CRBSL_INTD	= 00000024		
CRBSL_TOUTROUT	= 0000001C		
CRING_FULL	= 00000FC8	R	03
DATAPATH_730	= 00000575	R	03
DATAPATH_750	= 00000565	R	03
DATAPATH_780	= 00000583	R	03
DATAPATH_790	= 00000583	R	03
DATAPATH_8SS	= 00000555	R	03
DCS_BUS	= 00000080		
DDBSL_DDT	= 0000000C		
DDBST_NAME	= 00000014		
DEVSM_AVL	*****	X	02
DEVSM_ELG	*****	X	02
DEVSM_IDV	*****	X	02
DEVSM_ODV	*****	X	02
DEVSM_SHR	*****	X	02
DEVTYPE_ARRAY	= 000007DD	R	03
DISK CONID	= 00000000		
DPTSC_LENGTH	= 00000038		
DPTSC_VERSION	= 00000004		
DPT\$INITAB	= 00000038	R	02
DPTSM_NOUNLOAD	= 00000004		
DPTSM_SCS	= 00000008		
DPTSM_SVP	= 00000002		
DPT\$REINITAB	= 00000047	R	02
DPT\$TAB	= 00000000	R	02
DT\$_LESI	= 00000005		
DT\$_QDA50	= 0000000B		
DT\$_RDRX	= 00000007		
DT\$_TK50P	= 00000008		
DT\$_TU81P	= 00000006		
DT\$_UDA50	= 00000003		
DT\$_UDA50A	= 00000004		
DT\$_UQPORT	= 00000003		
DUP_CONID	= 00000002		
DYN\$C_CRB	= 00000005		
DYN\$C_DDB	= 00000006		
DYN\$C_DPT	= 0000001E		
DYN\$C_SCS	= 00000060		
DYN\$C_SCS_PB	= 00000004		
DYN\$C_SCS_PDT	= 00000005		
DYN\$C_SCS_SB	= 00000007		
DYN\$C_UCB	= 00000010		
EMB\$C_UM	= 00000004		
EMB\$C_DV_REGS AV	= 0000000E		
ENABLE_COMMAND_CODE	= 00000FD1	R	03
ENABLE_COMMAND_END	= 00000FB7	R	03

PUDRIVER
Symbol table

J 15

16-SEP-1984 01:05:05 VAX/VMS Macro V04-00
5-SEP-1984 00:17:10 [DRIVER.SRC]PUDRIVER.MAR;1

Page 98
(5)

```

ENABLE_COMMAND_OFFSET      = 0000001E
ENABLE_COMMAND_START      = 00000FB0 R 03
ENABLE_RESPONSE_CODE      = 00001350 R 03
ENABLE_RESPONSE_END      = 000012C2 R 03
ENABLE_RESPONSE_OFFSET    = 00000090
ENABLE_RESPONSE_START    = 000012BD R 03
ERL$DEVICEATTN          ***** X 03
ERL$LOGMESSAGE          ***** X 03
EXESALONONPAGED          ***** X 03
EXESALOPHYCNTG           ***** X 03
EXESGB_CPUTYPE           ***** X 03
EXESGL_ABSTIM            ***** X 03
EXESGL_TENUSEC           ***** X 03
EXESGL_UBDELAY           ***** X 03
EXESGQ_SYSTIME           ***** X 03
EXESIOFORK               ***** X 03
EXESSNDEVMSG             ***** X 03
EXESZEROPARM             ***** X 03
FAIL_ATTNCODE            = 00000002
FPC$ACCEPT               00000CFB RG 03
FPC$ALLOCDG              00000CFB RG 03
FPC$ALLOCMG              00000E2E RG 03
FPC$CONNECT              00000D21 RG 03
FPC$DCONNECT             00000DF1 RG 03
FPC$DEALLOCDG            00000CFB RG 03
FPC$DEALLOMSG            00000E95 RG 03
FPC$DEALRGMG             00000E90 RG 03
FPC$MAINTFCN             00000CFB RG 03
FPC$MAP                  00000CFB RG 03
FPC$MAPBYPASS            00000CFB RG 03
FPC$MAPIRP              00000FDE RG 03
FPC$MAPIRPBYP            00000CFB RG 03
FPC$MAPIRP_BDA           0000112F RG 03
FPC$MAPIRP_UV1           00001014 RG 03
FPC$MRESET               00000CFF RG 03
FPC$MSTART               00000DOB RG 03
FPC$QUEUEDG              00000E9D RG 03
FPC$QUEUEMDGS            00000CFB RG 03
FPC$RCHMSGBUF            00000EF3 RG 03
FPC$RCLMSGBUF            00000F32 RG 03
FPC$READCOUNT           00000CFB RG 03
FPC$REJECT               00000CFB RG 03
FPC$REQDATA              00000CFB RG 03
FPC$RLSCOUNT             00000CFB RG 03
FPC$SENDDATA             00000CFB RG 03
FPC$SENDDG              00000CFB RG 03
FPC$SENDMSG              00000CFB RG 03
FPC$SENDRGDG             00000CFB RG 03
FPC$SNDCNTMSG            00000F71 RG 03
FPC$STOP_VCS             00000CFB RG 03
FPC$UNMAP                00001130 RG 03
FPC$UNMAP_BDA            000011BD RG 03
FPC$UNMAP_UV1            00001162 RG 03
FUNCTAB_LEN              = 0000001C
GOTO_HARDP               000009C9 R 03
HARDPOWER                 00000AAF R 03
HARDWARE_INIT            00000811 R 03

```

```

HARD_RETRY                00000AB2 R 03
HS1_M_BIT15               = 00008000
HS1_M_IE                  = 00000080
HS1_V_CRNGLEN             = 0000000B
HS1_V_RRNGLEN             = 00000008
HS2_M_PI                  = 00000001
HS4_M_GO                  = 00000001
HS4_M_LF                  = 00000002
HS4_V_BURST               = 00000002
IDBSL_ADG                 = 00000014
IDBSL_CSR                 = 00000000
IDBSL_OWNER               = 00000004
ILLIOFUNC                 00000080 R 03
INISBRK                   ***** X 03
INIT_ATTNCODE             = 00000001
INIT_BDA_BUFS             000006CF R 03
INIT_CTLR                 0000022D RG 03
INIT_DELTA                = 0000000A
INIT_INIT_STEPS           00000701 R 03
INIT_PU_PDT               00000B28 R 03
INIT_UDA_BUFFERS          000004E5 R 03
INIT_UDA_BUFS_8SS         00000504 R 03
INIT_UDA_BUFS_RTN         000006D3 R 03
INIT_UDA_BUFS_UV1         000005E2 R 03
INSERT_IN_CRING           00000F9E R 03
INSERT_IN_FREEQ           00000ECE R 03
INSERT_IN_RRING           00000EA4 R 03
INSERT_RTN                00000ECD R 03
INTR_VEC                  = 000000B8
IOS_INITIALIZE             = 00000004
IOS_STOP                  = 00000003
IOS_VIRTUAL                = 0000003F
IOCSALLOSPT               ***** X 03
IOCSALLOUBMAPRM           ***** X 03
IOCSLOADUBAMAP            ***** X 03
IOCSLUBAUDAMAP            ***** X 03
IOCSMNTVER                ***** X 03
IOCSPURGDATAP             ***** X 03
IOCSRELDATAPUDA           ***** X 03
IOCSRELMAUDA              ***** X 03
IOCSREQCOM                ***** X 03
IOCSREQDATAP              ***** X 03
IOCSREQDATAPUDA           ***** X 03
IOCSREQMAPUDA             ***** X 03
IOCSREQPCANL              ***** X 03
IOCSRETURN                ***** X 03
IOCSTHREADCRB             ***** X 03
IOCSWFIKPCCH              ***** X 03
IPLS_SCS                  = 00000008
IRPS$FCODE                = 00000006
IRPSV_FCODE               = 00000000
IRPSV_FUNC                = 00000001
IRPSW_FUNC                = 00000020
LESI_CNTRLTYP             = 00000001
LOADBA_8SS                00000874 R 03
LOADUBA_COMMON            00000883 R 03
LOOP_LIMIT                = 00000FAB

```


PUDRIVER
Symbol table

K 15

16-SEP-1984 01:05:05 VAX/VMS Macro V04-00
5-SEP-1984 00:17:10 [DRIVER.SRC]PUDRIVER.MAR;1

Page 99
(5)

MAPSM_MAPREGS	= 00000001		
MAPSV_MAPREGS	= 00000000		
MAP_ODD	0000101B	R	03
MAP_UNALIGN	0000104B	R	03
MASKH	= 00000000		
MASKL	= 00000018		
MAYA_CNTRLTYP	= 00000003		
MMG\$GL_SPTBASE	*****	X	03
MSG\$_RC25MVER	= 0000005D		
MSG\$_RDRXMVER	= 0000005E		
MSG\$_TU81MVER	= 0000005F		
MSG\$_UDA50MVER	= 00000057		
NOLOADUBA_UV1	0000088F	R	03
NO_CONSEC_INITS	= 00000005		
NO_OVERLAYMAP	000003F0	R	03
NO_PHYCONTIGBYT	= 00000200		
NO_PHYCONTIGPGS	= 00000001		
NUCL_CDT	000000FC	R	03
NULL_DG_INPUT	000000E0	R	03
NULL_ERR_ROUT	000000F9	R	03
NULL_MSG_INPUT	000000DC	R	03
NULL_ROUTINE	000000F9	R	03
NUMUBAVEC	= 00000080		
OPC_MSG_ARRAY	00000804	R	03
OVERLAYMAP	000003E4	R	03
OVERLAYMAP_8SS	000003CB	R	03
PBSB_SUBTYP	= 0000000B		
PBSB_TYPE	= 0000000A		
PBSC_OPEN	= 00000003		
PBSK_LENGTH	= 00000054		
PBSL_BLINK	= 00000004		
PBSL_FLINK	= 00000000		
PBSL_PDT	= 0000002C		
PBSL_RPORT_REV	= 00000018		
PBSL_RPORT_TYP	= 00000014		
PBSL_SBLINK	= 00000030		
PBSL_WAITQBL	= 0000003C		
PBSL_WAITQFL	= 00000038		
PBST_LPORT_NAME	= 00000024		
PBSW_SIZE	= 00000008		
PBSW_STATE	= 00000012		
PB_ALLOC_FAIL	00000C68	R	03
PDT\$B_BDPUSECNT	0000012A		
PDT\$B_CONBITMAP	00000127		
PDT\$B_CPOLLINX	00000121		
PDT\$B_CRCONTENT	0000012C		
PDT\$B_CRINGCNT	00000122		
PDT\$B_CRINGINX	00000120		
PDT\$B_DATAPATH	00000129		
PDT\$B_NOCURCON	00000126		
PDT\$B_PDT_TYPE	= 00000007		
PDT\$B_PURGEDP	00000203		
PDT\$B_RPOLLINX	00000124		
PDT\$B_RRCONTENT	0000013C		
PDT\$B_RRINGCNT	00000125		
PDT\$B_RRINGINX	00000123		
PDT\$B_SERVERS	00000128		

PDT\$B_SUBTYP	= 0000000B
PDT\$B_TYPE	= 0000000A
PDT\$C_COMAREALN	= 00000088
PDT\$C_LENGTH	= 000000E4
PDT\$C_PU	= 00000002
PDT\$C_PULENGTH	= 00000C94
PDT\$C_RINGLEN	= 00000A88
PDT\$C_SCSBASE	= 0000000C
PDT\$C_SCSSEND	= 00000084
PDT\$C_UVILENGTH	= 00000EB4
PDT\$SL_ACCEPT	= 0000000C
PDT\$SL_ADP	= 000000E0
PDT\$SL_ALLOCDG	= 00000010
PDT\$SL_ALLOCMSG	= 00000014
PDT\$SL_BDTABLE	0000014C
PDT\$SL_CMDRING	00000248
PDT\$SL_COMAREA	00000200
PDT\$SL_CONNECT	= 00000018
PDT\$SL_DCONNECT	= 00000028
PDT\$SL_DEALLOCDG	= 0000001C
PDT\$SL_DEALLOMSG	= 00000020
PDT\$SL_DEALRGMSG	= 00000024
PDT\$SL_DGOVRHD	= 000000B8
PDT\$SL_MAINTFCN	= 00000078
PDT\$SL_MAP	= 0000002C
PDT\$SL_MAPBYPASS	= 00000030
PDT\$SL_MAPIRP	= 00000034
PDT\$SL_MAPIRPBYP	= 00000038
PDT\$SL_MAXBCNT	= 000000BC
PDT\$SL_MRESET	= 00000070
PDT\$SL_MSGHDRSZ	= 000000B4
PDT\$SL_MSTART	= 00000074
PDT\$SL_PQ_ALGNPG	00000CB4
PDT\$SL_PQ_MAP	00000C94
PDT\$SL_PQ_PGPHAD	00000CA4
PDT\$SL_PQ_PGQBL	00000C9C
PDT\$SL_PQ_PGQFL	00000C98
PDT\$SL_PQ_POWNER	00000CA0
PDT\$SL_PQ_SVPTE	00000CAC
PDT\$SL_PQ_UBFSVA	00000CA8
PDT\$SL_PQ_USRPT	00000CB0
PDT\$SL_PU_BUFARY	00000288
PDT\$SL_PU_BUFQBL	0000011C
PDT\$SL_PU_BUFQFL	00000118
PDT\$SL_PU_CDTARY	000000E4
PDT\$SL_PU_CSR	00000100
PDT\$SL_PU_FQBL	0000010C
PDT\$SL_PU_FQFL	00000108
PDT\$SL_PU_FQPTR	00000104
PDT\$SL_PU_SB	000000FC
PDT\$SL_PU_SNDQBL	00000114
PDT\$SL_PU_SNDQFL	00000110
PDT\$SL_PU_VCO	000000E4
PDT\$SL_PU_VC1	000000E8
PDT\$SL_PU_VC2	000000EC
PDT\$SL_PU_VC255	000000F0
PDT\$SL_QUEUEDG	= 0000003C

PUDRIVER
Symbol table

L 15

16-SEP-1984 01:05:05 VAX/VMS Macro V04-00
5-SEP-1984 00:17:10 [DRIVER.SRC]PUDRIVER.MAR;1

Page 100
(5)

PDT\$\$_QUEU\$MDGS	= 000000040		
PDT\$\$_RCHMSGBUF	= 000000044		
PDT\$\$_RCLMSGBUF	= 000000048		
PDT\$\$_READCOUNT	= 000000068		
PDT\$\$_REJECT	= 00000004C		
PDT\$\$_REQDATA	= 000000050		
PDT\$\$_RINGBASE	= 000000208		
PDT\$\$_RLSCOUNT	= 00000006C		
PDT\$\$_RSPRING	= 000000208		
PDT\$\$_SENDDATA	= 000000054		
PDT\$\$_SENDDG	= 000000058		
PDT\$\$_SENDMSG	= 00000005C		
PDT\$\$_SENDRGDG	= 00000007C		
PDT\$\$_SND\$CNTMSG	= 000000060		
PDT\$\$_STOP_VCS	= 000000080		
PDT\$\$_TRTABLE	= 000000C88		
PDT\$\$_TRTBLEND	= 000000C90		
PDT\$\$_TRTBLPTR	= 000000C8C		
PDT\$\$_UCBO	= 0000000DC		
PDT\$\$_UNMAP	= 000000064		
PDT\$\$_WAITQBL	= 0000000B0		
PDT\$\$_WAITQFL	= 0000000AC		
PDT\$\$_SNG\$HOST	= 000000001		
PDT\$\$_CMDINT	= 000000204		
PDT\$\$_PORTCHAR	= 000000004		
PDT\$\$_PU_CRD255	= 0000000FA		
PDT\$\$_PU_CRDARY	= 0000000F4		
PDT\$\$_PU_CRED0	= 0000000F4		
PDT\$\$_PU_CRED1	= 0000000F6		
PDT\$\$_PU_CRED2	= 0000000F8		
PDT\$\$_RSPINT	= 000000206		
PDT\$\$_SIZE	= 000000008		
POLL_CMDRING	= 000011BE	R	03
POLL_RSPRING	= 0000125D	R R	03
POST_POWER_FORK	= 000002C3	R R	03
POWER_INIT	= 000002B5	R	03
PR\$_IPL	= 000000012		
PR\$_SID_TYP730	= 000000003		
PR\$_SID_TYP750	= 000000002		
PR\$_SID_TYP780	= 000000001		
PR\$_SID_TYP790	= 000000004		
PR\$_SID_TYP8SS	= 000000005		
PR\$_SID_TYPUV1	= 000000007		
PR\$_TBIS	= 00000003A		
PS1_V_ER	= 00000000F		
PS1_V_OD	= 000000007		
PS1_V_S1	= 00000000B		
PS2_S_PORTTYPE	= 000000003		
PS2_V_ER	= 00000000F		
PS2_V_PORTTYPE	= 000000008		
PS2_V_S2	= 00000000C		
PS3_V_ER	= 00000000F		
PS3_V_S3	= 00000000D		
PS4_S_CNTRLTYP	= 000000007		
PS4_S_UCODEVER	= 000000004		
PS4_V_CNTRLTYP	= 000000004		
PS4_V_ER	= 00000000F		

PS4_V_S4	= 00000000E		
PS4_V_UCODEVER	= 000000000		
PTESC_KOWN	= 000000000		
PTESC_KW	= 100000000		
PTESM_PFN	= 001FFFFFF		
PTESM_VALID	= 800000000		
PTES\$\$_P\$FN	= 000000015		
PTESV_P\$FN	= 000000000		
PUSCT\$\$_INIT	= 00000022C	RG	03
PUSDDT	= 000000000	RG	03
PUSEND	= 00001395	R	03
PUS\$FUNCTABLE	= 0000000B2	R	03
PUSINT	= 00001201	RG	03
PUSL_DRIVER_STS	= 0000000D0	R	03
PUSL_MPHYAD	= 0000000D4	R	03
PUSL_TRACE_VARIABLE	= 0000000D8	R	03
PUS\$A_POLL	= 0000135B	R	03
PUS\$SC\$OFFSET	= 000001E4	RG	03
PUS\$STARTIO	= 000000038	R	03
PUS\$UNITINIT	= 000000222	RG	03
PURGE_730	= 000007A4	R	03
PURGE_750	= 000007A4	R	03
PURGE_780	= 000007A1	R	03
PURGE_790	= 000007A1	R	03
PURGE_8SS	= 000007A4	R	03
PURGE_ATTNCODE	= 000000004		
PURGE_UV1	= 000007A4	R	03
PU_REGDUMP	= 00000008D	R	03
QDA\$O_CNTRLTYP	= 00000000D		
Q_DEA\$LOC_BUF	= 000000E9D	R	03
RDRX_CNTRLTYP	= 000000007		
REG\$AVE	= 00000001C		
REL\$DATAPATH_TV	= 0000001A0	R	03
REL\$DATAP_730	= 0000001E3	R	03
REL\$DATAP_750	= 0000001CC	R	03
REL\$DATAP_8SS	= 0000001D7	R	03
REQ\$DATAPATH_TV	= 00000019C	R	03
REQ\$DATAP_730	= 0000001E3	R	03
REQ\$DATAP_750	= 0000001A4	R	03
REQ\$DATAP_8SS	= 0000001AF	R	03
RINGBASE_8SS	= 000000752	R	03
RINGBASE_COM	= 000000761	R	03
RINGBASE_UV1	= 000000775	R	03
SA_POLL_INTVAL	= 00000000F		
SB\$B_SYSTEMID	= 000000018		
SB\$B_TYPE	= 00000000A		
SB\$K_LENGTH	= 000000060		
SB\$\$_CSB	= 00000005C		
SB\$\$_P\$BBL	= 000000010		
SB\$\$_P\$B\$CONNX	= 000000014		
SB\$\$_P\$B\$FL	= 00000000C		
SB\$Q_SWINCARN	= 00000002C		
SB\$W_SIZE	= 000000008		
SB_ACLOC_FAIL	= 000000C68	R	03
SC\$S\$DEALC_CDT	= *****	X	03
SC\$S\$GB_UDABURST	= *****	X	03
SC\$S\$GQ_CONFIG	= *****	X	03

PUDRIVER
Symbol table

M 15

16-SEP-1984 01:05:05 VAX/VMS Macro V04-00
5-SEP-1984 00:17:10 [DRIVER.SRC]PUDRIVER.MAR;1

Page 101
(5)

SCS\$RESUMWAITR	*****	X	03
SCSCMG\$SCSCMGDEF	= 00000030		
SEND_OPC_MSG	00000B13	R	03
SERVERS_ARRAY	000007EA	R	03
SETUP_COPY_SEG1	000010A6	R	03
SETUP_COPY_SEG2	0000110F	R	03
SIZ...	= 00000001		
SS\$ILLCDTST	= 00002154		
SS\$ILLIOFUNC	= 000000F4		
SS\$NOLISTENER	= 0000215C		
SS\$NORMAL	= 00000001		
SS\$NOSUCHNODE	= 0000028C		
SS\$REJECT	= 00000294		
START_STOP	00000063	R	03
STATE_ERR	000011FB	R	03
STEP1_LIMIT	= 0000000A		
STEP1_TIMEOUT	00000AD9	R	03
STEP2_LIMIT	= 0000000A		
STEP2_TIMEOUT	00000AD9	R	03
STEP3_LIMIT	= 0000000A		
STEP3_TIMEOUT	00000AD9	R	03
STOCK_RSPRING	00000BAA	R	03
SUCCESS	0000007B	R	03
SY\$GL_OPRMBX	*****	X	03
TAPE_CONID	= 00000001		
TESTUDA_780	00000A69	R	03
TESTUDA_790	00000A69	R	03
TRACE_COMMAND	00000476	R	03
TRACE_COMMON	00000484	R	03
TRACE_RESPONSE	0000047E	R	03
TSTVC2	00000D56	R	03
TU81_CNTRLTYP	= 00000005		
UBMD\$B_DATAPATH	= 00000003		
UBMD\$W_MAPREG	= 00000000		
UCB\$B_DEVCLASS	= 00000040		
UCB\$B_DEVTYP	= 00000041		
UCB\$B_DIPL	= 0000005E		
UCB\$B_FIPL	= 0000000B		
UCB\$B_INITCNT	000000A5		
UCB\$B_TYPE	= 0000000A		
UCB\$B_UDAFLAGS	000000A4		
UCB\$C_PUSIZE	= 000000C4		
UCB\$L_CRB	= 00000024		
UCB\$L_DDB	= 00000028		
UCB\$L_DDT	= 00000088		
UCB\$L_DEVCHAR	= 00000038		
UCB\$L_DPC	= 0000009C		
UCB\$L_FPC	= 0000000C		
UCB\$L_FR3	= 00000010		
UCB\$L_FR4	= 00000014		
UCB\$L_PDT	= 00000084		
UCB\$L_PU_ALLOC	000000A0		
UCB\$L_PU_SVAPTE	000000BC		
UCB\$L_SVAPTE	= 00000078		
UCB\$L_SVPN	= 00000074		
UCB\$M_ONLINE	= 00000010		
UCB\$M_POWER	= 00000020		

UCB\$M_PU_FRKBSY	= 00000002		
UCB\$M_PU_HRDINI	= 00000004		
UCB\$M_PU_INILOG	= 00000040		
UCB\$M_PU_MRESET	= 00000010		
UCB\$V_INT	= 00000001		
UCB\$V_ONLINE	= 00000004		
UCB\$V_POWER	= 00000005		
UCB\$V_PU_BDPATH	= 00000005		
UCB\$V_PU_FRKBSY	= 00000001		
UCB\$V_PU_INILOG	= 00000006		
UCB\$V_PU_MRESET	= 00000004		
UCB\$W_ATTNCODE	000000A8		
UCB\$W_BCNT	= 0000007E		
UCB\$W_BOFF	= 0000007C		
UCB\$W_DEVSTS	= 00000068		
UCB\$W_HOSTSTEP1	000000AE		
UCB\$W_HOSTSTEP2	000000B2		
UCB\$W_HOSTSTEP3	000000B6		
UCB\$W_HOSTSTEP4	000000BA		
UCB\$W_NUMBINIT	000000A6		
UCB\$W_PORTSTEP1	000000AC		
UCB\$W_PORTSTEP2	000000B0		
UCB\$W_PORTSTEP3	000000B4		
UCB\$W_PORTSTEP4	000000B8		
UCB\$W_PU_BCNT	000000C2		
UCB\$W_PU_BOFF	000000C0		
UCB\$W_STS	= 00000064		
UCB\$W_UDASA	000000AA		
UCB\$W_UNIT	= 00000054		
UCODE_ATTNCODE	= 00000005		
UCODE_VER_ARRAY	000007F7	R	03
UDASK_CREDTYPE	= 00000002		
UDASK_DGTYPE	= 00000001		
UDASK_MAINTTYPE	= 0000000F		
UDASK_RINGEXP	= 00000004		
UDASK_RINGSIZE	= 00000010		
UDASK_SEQMSGTYP	= 00000000		
UDAS\$M_STOPPED	= 00000002		
UDAS\$V_STOPPED	= 00000001		
UDA50A_CNTRLTYP	= 00000006		
UDA50_CNTRLTYP	= 00000000		
UDA730	00000A8F	R	03
UDA750	00000A74	R	03
UDA8SS	00000A74	R	03
UDAB\$B_BUFFNO	0000000A		
UDAB\$B_CONID	00000013		
UDAB\$B_CREDTYPE	00000012		
UDAB\$B_RINGINX	00000008		
UDAB\$B_RINGNO	00000009		
UDAB\$C_LENGTH	= 00000050		
UDAB\$L_BLINK	00000004		
UDAB\$L_CTRLHDR	00000010		
UDAB\$L_DESCRIP	0000000C		
UDAB\$L_FLINK	00000000		
UDAB\$S_CREDITS	= 00000004		
UDAB\$S_MSGTYPE	= 00000004		
UDAB\$T_TEXT	00000014		

PUDRIVER
Symbol table

N 15

16-SEP-1984 01:05:05 VAX/VMS Macro V04-00
5-SEP-1984 00:17:10 [DRIVER.SRC]PUDRIVER.MAR;1

Page 102
(5)

UDABSV_CREDITS	=	00000000		
UDABSV_MSGTYPE	=	00000004		
UDABSW_MSG_LEN		00000010		
UDAIP		00000000		
UDASA		00000002		
UDASA_ATTNCODE	=	00000003		
UDAUVT		00000A8F	R	03
UDA_M_FLAG	=	40000000		
UDA_M_OWN	=	80000000		
UDA_OUTFREV		00000ADF	R	03
UDA_V_FLAG	=	0000001E		
UDA_V_OWN	=	0000001F		
UNMAP_ODD		00001166	R	03
UNMAP_UNALIGN		00001170	R	03
UPDATE_PB_SB		00000C6B	R	03
UQPORT_M_MAPPED	=	80000000		
UV1_PDT_LENGTH	=	00001000		
UV1_PDT_PAGES	=	00000008		
VASS_VPN	=	00000015		
VASV_VPN	=	00000009		
VCONAM		00000D0C	R	03
VCONAMLEN	=	00000009		
VC1NAM		00000D15	R	03
VC1NAMLEN	=	00000009		
VC2NAM		00000D1E	R	03
VC2NAMLEN	=	00000003		
VECSB_DATAPATH	=	00000013		
VECSB_NUMREG	=	00000012		
VECSL_ADP	=	00000014		
VECSL_IDB	=	00000008		
VECSL_INITIAL	=	0000000C		
VECSL_UNITINIT	=	00000018		
VECSW_MAPREG	=	00000010		
VIRT_TO_PHYAD		000006D4	R	03
WAITTODS		00000AF4	R	03

+-----+
! Psect synopsis !
+-----+

PSECT name	Allocation	PSECT No.	Attributes														
. ABS .	00000000 (0.)	00 (0.)	NOPIC USR	CON	ABS	LCL	NOSHR	NOEXE	NORD	NOWRT	NOVEC	BYTE					
\$ABSS	00000EB4 (3764.)	01 (1.)	NOPIC USR	CON	ABS	LCL	NOSHR	EXE	RD	WRT	NOVEC	BYTE					
\$\$\$105_PROLOGUE	0000005C (92.)	02 (2.)	NOPIC USR	CON	REL	LCL	NOSHR	EXE	RD	WRT	NOVEC	BYTE					
\$\$\$115_DRIVER	00001395 (5013.)	03 (3.)	NOPIC USR	CON	REL	LCL	NOSHR	EXE	RD	WRT	NOVEC	LONG					

+-----+
! Performance indicators !
+-----+

Phase	Page faults	CPU Time	Elapsed Time
Initialization	36	00:00:00.03	00:00:02.71
Command processing	136	00:00:00.43	00:00:03.31
Pass 1	1207	00:00:33.14	00:01:59.58
Symbol table sort	0	00:00:03.55	00:00:12.38

Pass 2	467	00:00:08.49	00:00:30.27
Symbol table output	2	00:00:00.33	00:00:00.86
Psect synopsis output	2	00:00:00.01	00:00:00.01
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	1853	00:00:45.99	00:02:49.14

The working set limit was 2850 pages.

266333 bytes (521 pages) of virtual memory were used to buffer the intermediate code.

There were 170 pages of symbol table space allocated to hold 3129 non-local and 179 local symbols.

4287 source lines were read in Pass 1, producing 32 object records in Pass 2.

77 pages of virtual memory were used to define 71 macros.

+-----+
! Macro library statistics !
+-----+

Macro library name	Macros defined
-----	-----
_\$255\$DUA28:[DRIVER.OBJ]PALIB.MLB;1	3
-\$255\$DUA28:[SYS.OBJ]LIB.MLB;1	46
-\$255\$DUA28:[SYSLIB]STARLET.MLB;2	11
TOTALS (all libraries)	60

3247 GETS were required to define 60 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LIS\$:PUDRIVER/OBJ=OBJ\$:PUDRIVER MSRC\$:PUDRIVER/UPDATE=(ENH\$:PUDRIVER)+EXECML\$/LIB+LIB\$:PALIB.MLB/LIB

0115 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

